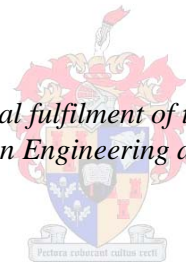


# **Dynamic Reconfigurable Platform for Swarm Robotics**

by

Gerhardus Heath

*Thesis presented in partial fulfilment of the requirements for the degree  
Master of Science in Engineering at Stellenbosch University*



Supervisor: Mr Willem Smit  
Department of Electrical and Electronic Engineering

March 2011

# **DECLARATION**

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: March 2011

Copyright © 2011 StellenboschUniversity

All rights reserved.

## **Abstract**

Swarm intelligence research was inspired by biological systems in nature. Working ants and bees has captivated researchers for centuries, with the ant playing a major role in shaping the future of robotic swarm applications. The ants foraging activity can be adapted for different applications of robotic swarm intelligence. Numerous researchers have conducted theoretical analysis and experiments on the ants foraging activities and communication styles.

Combining this information with modern reconfigurable computing opens the door to more complex behaviour with improved system dynamics. Reconfigurable computing has numerous applications in swarm intelligence such as true hardware parallel processing, dynamic power save algorithms and dynamic peripheral changes to the CPU core.

In this research a brief study is made of swarm intelligence and its applications. The ants' foraging activities were studied in greater detail with the emphasis on a layered control system designed implementation in a robotic agent. The robotic agent's hardware was designed using a partial self reconfigurable FPGA as the main building element. The hardware was designed with the emphasis on system flexibility for swarm application drawing attention to power reduction and battery life. All of this was packaged into a differential drive chassis designed specifically for this project.

## Opsomming

Die motivering vir swerm robotika kom van die natuur. Vir eeue fassineer swerm insekte soos bye en miere navorsers. Dit is verstommend hoe 'n groep klein en nietige insekte sulke groot take kan verrig. Die mier speel 'n belangrike rol en is die sentrale tema van menige publikasies. Die mier se kos-soek aktiwiteit kan aangepas word vir swerm robotika toepassings. Hierdie aktiwiteit vervat verskeie sleutel konsepte wat belangrik is vir robotika toepassings.

Deur bv. die mier se aktiwiteite te kombineer met dinamies herkonfigureerbare hardeware, kan meer komplekse gedrag bestudeer word. Die stelsel dinamika verbeter ook, aangesien dit nou moontlik is om sekere take in parallel uit te voer. Deur 'n interne prosesseerder in die herkonfigureerbare hardeware in te sluit, is dit nou vir die stelsel moontlik om homself te verander tydens taak verrigting. Komplekse krag bestuur gedrag is ook moontlik deurdat die prosesseerder die spoed en rand apparaat kan verander soos benodig. 'n Verdere voordeel is dat die stelsel aanpasbaar is en dus vir verskeie navorsingsprojekte gebruik kan word.

In hierdie navorsing word 'n literatuur studie van swerm robotika gemaak en word daar ook na toepassings gekyk. Met die klem op praktiese implementering, word die mier se kos-soek aktiwiteit in detail ondersoek deur gebruik te maak van 'n laag beheerstelsel. In hierdie laag beheerstelsel verteenwoordig elke laag 'n hoër vlak gedrag. Stelsel aanpasbaarheid en lae kragverbruik speel 'n deurslaggewende rol in die ontwerp, en om hierdie rede vorm 'n FPGA die hart van die sisteem.

## Acknowledgements

Special thanks to:

- My lord Jesus Christ for giving me this opportunity. I am truly blest and He has opened many doors for me.
- My wife and children for their support, while I spent countless evenings and weekends doing research.
- Willem Smit and Johan Treurnicht for always assisting me in a friendly and helpful manner. Their guidance and financial assistance enabled me to conduct this research.
- Xilinx who sponsored all the software tools used for my research.

# Table of content

<b>DECLARATION .....</b>	<b>ii</b>
<b>Abstract .....</b>	<b>iii</b>
<b>Opsomming .....</b>	<b>iv</b>
<b>Acknowledgements .....</b>	<b>v</b>
<b>Table of content.....</b>	<b>2</b>
<b>List of Figures .....</b>	<b>5</b>
<b>List of tables .....</b>	<b>8</b>
<b>Abbreviations .....</b>	<b>9</b>
<b>Chapter 1 : Introduction .....</b>	<b>10</b>
Section 1.1 Problem description.....	10
Section 1.2 Proposed solution. The birth of ASRA.....	12
Section 1.3 Outline.....	12
<b>Chapter 2 Swarm-Based Robotics .....</b>	<b>14</b>
Section 2.1 Biologic Inspiration .....	14
Section 2.1.1 Cooperation.....	15
Section 2.1.2 Communication .....	15
Section 2.1.3 Ant behaviour.....	16
Section 2.1.3.1 Foraging .....	16
Section 2.1.3.2 Learning .....	17
Section 2.2 Layered Control System (Subsumption).....	17
Section 2.2.1 Basis/Basic Behaviour .....	18
Section 2.2.1.1 Safe-wandering .....	19
Section 2.2.1.2 Following .....	20
Section 2.2.1.3 Dispersion.....	21
Section 2.2.1.4 Aggregation .....	22
Section 2.2.1.5 Homing.....	23
Section 2.2.2 Higher level behaviour. Combining Basic Behaviour .....	23
Section 2.2.3 Typical Higher level Behaviours.....	24
Section 2.2.3.1 Foraging .....	24

Section 2.2.3.2 Flocking.....	24
<b>Section 2.3 Swarm implementation of the Ant's Basic Behaviour .....</b>	<b>25</b>
Section 2.3.1 Experiment.....	27
Section 2.3.2 Weaknesses.....	28
<b>Chapter 3 : Reconfigurable Computing .....</b>	<b>29</b>
<b>Section 3.1 FPGA .....</b>	<b>29</b>
<b>Section 3.2 Dynamic Reconfiguration.....</b>	<b>31</b>
Section 3.2.1 Dynamic Partial Reconfiguration .....	31
Section 3.2.2 Dynamic Partial Self Reconfiguration .....	31
<b>Section 3.3 Xilinx architecture .....</b>	<b>32</b>
Section 3.3.1 Dynamic Partial Reconfiguration .....	32
Section 3.3.2 Dynamic Partial Self Reconfiguration .....	34
Section 3.3.3 Tools.....	35
<b>Section 3.4 FPGA Power considerations and reduction techniques .....</b>	<b>37</b>
Section 3.4.1 Static and dynamic consumption.....	37
Section 3.4.2 Clock Frequency .....	38
Section 3.4.3 Communication primitives.....	38
<b>Chapter 4 Design.....</b>	<b>40</b>
<b>Section 4.1 Main Board .....</b>	<b>41</b>
Section 4.1.1 FPGA selection .....	41
Section 4.1.2 Processor Soft Core .....	43
Section 4.1.2.1 LM32 soft-core architecture .....	45
Section 4.1.3 Peripherals .....	46
Section 4.1.4 HDL design.....	48
Section 4.1.4.1 Peripherals.....	50
Section 4.1.4.2 Clocks .....	57
Section 4.1.4.3 Power management.....	57
Section 4.1.4.4 Dynamic Partial Self Reconfiguration.....	58
Section 4.1.4.5 Floorplanning.....	58
Section 4.1.5 Power consumption optimisation .....	60
Section 4.1.6 Hardware Design.....	62
Section 4.1.6.1 Component selection .....	63
Section 4.1.6.2 Core temperature sensing .....	64
Section 4.1.6.3 Interfaces .....	64
Section 4.1.6.4 FPGA configuration.....	65
Section 4.1.6.5 PCB.....	66
Section 4.1.7 Software.....	72

Section 4.1.7.1 Debugger .....	72
Section 4.1.7.2 OS support.....	76
<b>Section 4.2 Power supply .....</b>	<b>77</b>
Section 4.2.1 Component selection .....	80
<b>Section 4.3 Peripheral Board .....</b>	<b>82</b>
Section 4.3.1 Component selection .....	83
Section 4.3.2 Design.....	87
<b>Section 4.4 Sensors .....</b>	<b>89</b>
Section 4.4.1 Proximity .....	89
Section 4.4.2 Sonar.....	93
Section 4.4.3 Shaft Encoder.....	96
Section 4.4.3.1 The circuit .....	98
<b>Section 4.5 Assembly and testing .....</b>	<b>99</b>
Section 4.5.1 Main board.....	99
Section 4.5.2 Power supply.....	104
Section 4.5.3 Peripheral board .....	106
<b><i>Chapter 5 Robot Construction.....</i></b>	<b><i>108</i></b>
<b>Section 5.1 Motor .....</b>	<b>112</b>
Section 5.1.1 Experimental results .....	117
<b><i>Chapter 6 Conclusion and future work.....</i></b>	<b><i>121</i></b>
<b>Section 6.1 Original contribution.....</b>	<b>121</b>
<b>Section 6.2 Known issues.....</b>	<b>122</b>
<b>Section 6.3 Future work .....</b>	<b>126</b>
<b><i>References .....</i></b>	<b><i>127</i></b>



# List of Figures

FIGURE 1. LAYERED CONTROL [2].....	18
FIGURE 2. ALGORITHMIC PSEUDO CODE FOR SAFE-WANDERING .....	20
FIGURE 3. ALGORITHMIC PSEUDO CODE FOR FOLLOWING .....	21
FIGURE 4. ALGORITHMIC PSEUDO CODE FOR DISPERSION .....	22
FIGURE 5. ALGORITHMIC PSEUDO CODE FOR AGGREGATION .....	22
FIGURE 6. ALGORITHMIC PSEUDO CODE FOR HOMING .....	23
FIGURE 7. BASIC BEHAVIOUR ARBITRATION FOR FORAGING .....	24
FIGURE 8. BASIC BEHAVIOUR ARBITRATION FOR FLOCKING.....	25
FIGURE 9. COMPETENCE LEVEL 0 [1:184] .....	26
FIGURE 10. COMPETENCE LEVEL 1 [1:184].....	26
FIGURE 11. COMPETENCE LEVEL 2 [1:185].....	26
FIGURE 12. REMAINING TARGETS VS. TIME FOR SINGLE CLUSTER OF TARGETS (1 CLUSTER) [1:205].....	27
FIGURE 13. COLLECTION TIME VERSUS NUMBER OF ROBOTS (1 CLUSTER) [1:206] .....	27
FIGURE 14. COLLISION FREQUENCY VERSUS NUMBER OF ROBOTS (1 CLUSTER) [1:207].....	28
FIGURE 15. LOGIC CELL .....	30
FIGURE 16. TYPICAL RECONFIGURABLE LOGIC FOR THE SPARTAN3 .....	33
FIGURE 17. VIRTEX4 LX15 FLOOR PLAN (ROTATED BY 90 DEG).....	33
FIGURE 18. SELF RECONFIGURATION BLOCK DIAGRAM.....	35
FIGURE 19. XILINX PLANAhead SOFTWARE .....	36
FIGURE 20. HIERARCHICAL INTERCONNECT RESOURCES [13:176] .....	39
FIGURE 21. ONE VIRTEX 4 CLB SHOWING ROUTING LINES .....	39
FIGURE 22. SYSTEM BLOCK DIAGRAM .....	40
FIGURE 23. VIRTEX FPGA CELL COST RATIO.....	42
FIGURE 24. VIRTEX FPGA BRAM COST RATIO .....	42
FIGURE 25. LM32 BLOCK DIAGRAM .....	45
FIGURE 26. WISHBONE READ AND WRITE BUS CYCLE .....	47
FIGURE 27. CPU ADDRESS MAP FOR DIFFERENT MODES .....	49
FIGURE 28. PROCESSOR & PERIPHERALS FLOORPLAN.....	59
FIGURE 29. LM32 PBlock AREAS AND FLOOR PLANNING.....	60
FIGURE 30. MAIN BOARD BLOCK DIAGRAM .....	62
FIGURE 31. CONFIGURATION CLOCK BOARD LAYOUT [33:34].....	67
FIGURE 32. PCB LAYER STACKUP .....	67
FIGURE 33. RECONFIGURATION CLOCK MEASURED AT CONFIGURATION PROM .....	68
FIGURE 34. CONFIGURATION PROM SELECTMAP LINE D1 .....	69
FIGURE 35. INSTRUCTION BUS LINE D8 AT 8mA.....	69
FIGURE 36. INSTRUCTION BUS LINE D8 AT 12mA.....	69
FIGURE 37. INSTRUCTION BUS LINE A7 AT 12mA.....	70
FIGURE 38. INSTRUCTION BUS TRACE A7 AT 8mA.....	70
FIGURE 39. INSTRUCTION BUS CROSS TALK .....	71

FIGURE 40. DATA BUS TRACE A7 AT 8mA .....	71
FIGURE 41. DATA BUS TRACE A7 AT 12mA .....	71
FIGURE 42. DATA BUS CROSS TALK.....	72
FIGURE 43. FREERTOS TASKS .....	77
FIGURE 44. PSU BLOCK DIAGRAM .....	80
FIGURE 45. SYNCHRONOUS CONVERTER EFFICIENCY .....	81
FIGURE 46. CONVERTER LOAD TRANSIENT RESPONSE .....	81
FIGURE 47. PERIPHERAL BOARD BLOCK DIAGRAM .....	83
FIGURE 48. ZIGBEE MESH NETWORK .....	86
FIGURE 49. DIGIMESH NETWORK [39] .....	87
FIGURE 50. H-BRIDGE BLOCK DIAGRAM.....	87
FIGURE 51. PROXIMITY SMART SENSOR SCHEMATIC.....	89
FIGURE 52. TSAL6200 INFRARED LED BEAM WIDTH .....	90
FIGURE 53. FIXED LENGTH BIT WORD FOR LOGIC 0 AND 1.....	91
FIGURE 54. PROTOTYPE CIRCUIT FOR PROXIMITY DETECTOR .....	92
FIGURE 55. PROXIMITY SENSOR OBJECT DETECTED.....	93
FIGURE 56. SONAR TRANSMITTER BLOCK DIAGRAM.....	93
FIGURE 57. SONAR RECEIVER BLOCK DIAGRAM .....	94
FIGURE 58. TYPICAL WALL RESPONSE. RANGE VS. ORIENTATION PLOT [41:37] .....	95
FIGURE 59. QUADRATURE DECODER STEPS .....	97
FIGURE 60. HUB ASSEMBLY SHOWING SHAFT ENCODER .....	97
FIGURE 61. SHAFT ENCODER PCB AND INSTALLATION .....	98
FIGURE 62. SHAFT ENCODER WAVEFORMS .....	98
FIGURE 63. TOP AND BOTTOM VIEW OF THE MAIN BOARD .....	99
FIGURE 64. POWER SUPPLY RISE TIME .....	100
FIGURE 65. FPGA POWER-UP [33:16].....	101
FIGURE 66. POWER-ON-RESET .....	101
FIGURE 67. VHDL TEST APPLICATION.....	104
FIGURE 68. TOP AND BOTTOM VIEW OF THE POWER SUPPLY BOARD .....	104
FIGURE 69. SUPPLY VOLTAGE RIPPLE. LEFT:2.5V (YELLOW) & 1.2V (GREEN). RIGHT: 5V (YELLOW) & 3.3V (GREEN).....	105
FIGURE 70. TOP AND BOTTOM VIEW OF THE PERIPHERAL BOARD .....	106
FIGURE 71. H-BRIDGE WAVEFORMS .....	106
FIGURE 72. ROBOT BOTTOM VIEW.....	108
FIGURE 73. BALL CASTER .....	109
FIGURE 74. ROBOT SIDE VIEW.....	109
FIGURE 75. ROBOT FRONT VIEW.....	110
FIGURE 76. ROBOT PERSPECTIVE VIEW .....	110
FIGURE 77. CONSTRUCTED ROBOTIC AGENT .....	112
FIGURE 78. DC MOTOR MODEL [37:205] .....	113

FIGURE 79. GEARED MOTOR .....	116
FIGURE 80. MOTOR NO LOAD AND STALL MEASUREMENTS.....	119
FIGURE 81. MOTOR POWER AND TORQUE GRAPHS.....	119
FIGURE 82. ADDITIONAL MOTOR MOUNTINGS.....	124

## List of tables

TABLE 1. VIRTEX4 LX15 CLOCK REGION DEFINITION .....	34
TABLE 2. SOFT CORES REVIEW .....	44
TABLE 3. WISHBONE SEL_O BYTE MAP .....	47
TABLE 4. WISHBONE DATA IN AND OUT BUS DEFINITION .....	48
TABLE 5. I/O PORT VHDL COMPONENT DEFINITION .....	50
TABLE 6. TIMER VHDL COMPONENT DEFINITION .....	51
TABLE 7. UART VHDL COMPONENT DEFINITION .....	52
TABLE 8. BAUD GENERATOR CODE .....	52
TABLE 9. ONBOARD ROM/RAM VHDL COMPONENT DEFINITION.....	53
TABLE 10. EXTERNAL MEMORY BUS VHDL COMPONENT DEFINITION.....	54
TABLE 11. SPI VHDL COMPONENT DEFINITION .....	55
TABLE 12. H-BRIDGE VHDL COMPONENT DEFINITION.....	56
TABLE 13. QUADRATURE DECODER VHDL COMPONENT DEFINITION .....	56
TABLE 14. FPGA POWER ANALYSIS WITHOUT POWER OPTIMISATION .....	60
TABLE 15. FPGA POWER ANALYSIS WITH POWER OPTIMISATION.....	61
TABLE 16. SOC POWER ANALYSIS BY HIERARCHY.....	61
TABLE 17. IO STANDARDS DC VOLTAGE SPECIFICATIONS AT VARIOUS VOLTAGE REFERENCES [30:258].....	68
TABLE 18. GDB-STUB CODE SEGMENT FOR STEP COMMAND .....	74
TABLE 19. FPGA SOFT-CORE EMULATED FLASH STORAGE DESCRIPTION.....	75
TABLE 20. CONFIGURATION FILE FOR DATA2MEM APPLICATION.....	76
TABLE 21. VIRTEX 4 QUIESCENT CURRENT .....	78
TABLE 22. RESET VOLTAGE THRESHOLD .....	79
TABLE 23. PSU MAXIMUM CURRENT DESIGN REQUIREMENT .....	79
TABLE 24. A2D COMPARISON .....	84
TABLE 25. ZIGBEE MODULES .....	86
TABLE 26. FPGA POWER SUPPLY RAMP TIME [31:7] .....	100
TABLE 27. GEARED MOTOR SPECIFICATIONS .....	115
TABLE 28. MOTOR RESISTANCE MEASUREMENTS.....	117

## Abbreviations

DCI	Digitally Controller Impedance
LC	Low Capacitance
DCM	Digital Clock Manager
PMCD	Phase-Matched Clock Divider
GCB	Global Clock Buffer
GC	Global Clock
ICAP	Internal Reconfiguration Access Port
RU	Reconfigurable Unit
HDL	Hardware Description Language
FPGA	Field Programmable Gate Array
LUT	Look Up Table
SOC	System On Chip
IOB	Input Output Block
CLB	Configurable Logic Block
BRAM	Block Random Access Memory
RU	Reconfigurable Unit
SO	Self Organisation
WASSO	Weighted Average Simultaneous Switching Output
PR	Partial Reconfiguration
LM32	LatticeMicro32
LDO	Low Drop Out
RTOS	Real Time Operating System
JTAG	Joint Test Action Group
SINAD	Signal-to-noise ration plus distortion
DPSR	Dynamic Partial Self Reconfiguration
RCD	Region of Constant Depth
UCF	User Constraint File
ASRA	Autonomous Swarm Robotic Agent
DOF	Degrees Of Freedom
IMU	Inertial Measurement Unit

# Chapter 1 : Introduction

## ***Section 1.1 Problem description***

Swarm intelligence is not a new field. MIT conducted research in this field as early as 1989. Since then authors like Rodney A Brooks and Maja J Matarić wrote numerous papers on the subject. Many of the earlier research conducted focused on theoretical analysis proven by simulation results. The basis for the research came from biological organisms. Later research activities included experimentation with physical agents. The number of agents used (typically 5 to 10) was far lower than those in the simulation models, but it yield similar results.

In most research the hardware were designed for a specific robot agent and was based on a fixed processor with some peripherals. After the design phase, a prototype was build, tested and any hardware related issues were addressed through wire modifications or a second design phase. Once the hardware was working, the specific research was conducted and the results published. This approach does not only contain risk in the design and testing phase, but also in the system implementation phase. During the design and testing phase, certain design limitations can only be overcome through additional design iteration. The same applies to the system implementation phase during which system complexity can result in too few available resources and/or inadequate processing power. In some of the research subsumption control was used which depends on parallel processing. Subsumption starts from the lowest level after which control layers are added to achieve more complex behaviour. With each layer added comes a higher demand on system resources. Inadequate system resources or incorrect design assumptions mandates a new hardware design phase which is time consuming and costly.

Another system requirement overlooked frequently is system power consumption and management. All swarm intelligent experiments considered in the literature study requires time for the swarm interaction to become apparent. Some of the studies showed an increase in error over time due to lost of speed as a result of battery drain.

In some cases the designer catered for a docking station to charge the agent but no intelligent power management is included or any form of battery energy storage tracking.

An autonomous robot agent is just as good as its sensor network. Most smaller agents contains an array of sensors comprising of micro switches with whiskers for close proximity sensing, Sharp IR distance sensors for short range sensing and low cost sonar for medium range sensing. The micro switch proximity barrier requires multiple switches and additional mechanical components. The complexity lies in the mechanical design to form a tactile like barrier around the robotic agent. The Sharp IR distance sensor family contains sensors with different sensing range. All of these sensors have a minimum sensing range and due to nonlinearities the minimum distance region is a very dangerous region as the sensor output rapidly falls and can be misinterpreted as an object further away. Low cost sonar can't sense within this range as its minimum range is even worse. Another problem with low cost sonar is it does threshold detection which can't be changed dynamically nor is it possible to determine the geometry of the reflected object. The sonar minimum sensing distance is a function of the threshold and the 40kHz transducers used. Unfortunately higher frequency transducers become more expensive.

For autonomous indoor navigation it is necessary to use dead reckoning at times since a GPS signal is not always present. Because dead reckoning is based on integration the error accumulates over time. Thus, to minimise the error it is important to use accurate encoders with high resolution. Accurate shaft encoders are expensive and therefore most of the implementations are based on optical sensing an encoded pattern, printed on a disk attached to the wheel. The accuracy and resolution is limited.

The problem description can thus be summarised under six points: 1) Hardware and system flexibility. 2) Power management for battery operated agents. 3) Sensor selection and refinement. 4) Mechanical design. 5) System design specifically for swarm intelligence application. 6) Make use of the open source community when deciding on tool chains and libraries/implementations. If possible use royalty free components.

## ***Section 1.2 Proposed solution. The birth of ASRA***

It is the aim of this research to develop a dynamic reconfigurable Autonomous Swarm Robotic Agent (ASRA) platform with its focus on swarm intelligence experiments. A phased approach is suggested: 1) develop a hardware platform for a robotic agent 2) develop a debug environment and port a real-time operating system for the hardware platform. 3) investigate battery cell technologies to choose appropriate battery cells 4) investigate appropriate sensors, choose the correct sensor complement and design additional sensors to overcome some of the limitations explained in the previous section. 5) Design the mechanical components to construct a robotic agent. Integrate the hardware platform, power management layer, sensors and batteries into a small three wheel differential drive robotic chassis.

System flexibility is the highest priority driving the design of the hardware. The hardware must be dynamic reconfigurable and it is therefore necessary to look at soft CPU cores implemented on a FPGA in a HDL language. A robotic agent will incorporate different sensor technologies with different interfaces, and the hardware should be able to accommodate the diverse interfaces. Inter-agent communication is also required to improve the collective behaviour of the swarm.

Partial reconfiguration will be used as a tool to improve system flexibility and limit the system power consumption through dynamically altering the System-On-Chip to scale the performance and available peripherals. Dynamic reconfiguration has its own unique requirements which are briefly discussed later on.

## ***Section 1.3 Outline***

Chapter 2 is an introduction into swarm robotics. This chapter also takes a look at the swarm intelligence of the ant and discuss an elegant control mythology for swarm applications. At the end of the chapter an ant swarm experiment is discussed.

Chapter 3 is a general discussion explaining key concepts in FPGA dynamic reconfiguration and self reconfiguration. This chapter also takes a closer look at the limitation and application of Xilinx technologies. At the end of the chapter some time is spend on current practical implementations.



Chapter 4 discuss the hardware design implementation to fulfil the requirements discussed in Section 1.1

Chapter 5 looks at the mechanical design and construction of the robotic agent. Motor calculations and selection is also discussed here.

Chapter 6 contains the conclusion of this report and looks at future work building on this design.

## Chapter 2 Swarm-Based Robotics

Swarm-based robotic research started in the early 1970's in the field of distributed artificial intelligence (DAI). The research was limited to software agents and it was only in the late 1980's that experimentation with physical agents started. In 1988 Gerardo Beni started working on the topic and formulated a vague definition of swarm intelligence: *"a property of systems of non-intelligent robots exhibiting collectively intelligent behaviour"* [21]. Deneubourg, Theraulaz, and Beckers studied swarm intelligence from an ethological perspective and they define a swarm as *"...a set of (mobile) agents which are liable to communicate directly or indirectly (by acting on their local environment) with each other, and which collectively carry out a distributed problem solving"* [22].

Swarm based robotics fall within the broader classification of cooperative, autonomous, mobile robotics. This research field focus on multiple agents working together to achieve a common goal. The main difference in comparison with AI is the lack of complexity and intelligence of a single agent. Traditionally AI robots were created to solve complex problems. These single agent solutions was advanced with high processing capabilities but still posed a single point of failure. A swarm robotic agent is simple, inexpensive and disposable. When an agent is faulty one of the other agents will take its place and collectively the task will be completed. On its own an agent cannot demonstrate complex behaviour but in a swarm the emerging behaviour is complex. Thus the true intelligence of swarm robotics is exposed through collective behaviour.

Multi agent robotic cooperation has the following benefits: 1) It can accomplish tasks that are often inherently too complex for a single agent. 2) Several simple robots can be cheaper than one powerful agent. 3) Multiple agents are more fault-tolerant than one single agent acting alone.

### **Section 2.1 Biologic Inspiration**

An insect is a complex creature, yet the complexity of an individual insect is not sufficient to explain complexities of what social insect colonies can do. Swarm insects do not have a leader yet each individual complete its task and collectively they can complete complex tasks.

### Section 2.1.1 Cooperation

Cooperation mainly arises through two mechanisms (see [23:1]) Genetic differences. For instance the anatomical differences between majors and minors in polymorphic species of ants can organise the division of labour. 2) Self organisation (SO). Camazine present the following definition of SO: *“Self-organization is a process in which patterns at the global level of a system emerge solely from numerous interactions among the lower-level components of the system. Moreover, the rules specifying interactions among the system’s components are executed using only local information, without reference to the global pattern”*[24:8]. Yet another definition from Bonabeau focuses more directly on ethological SO: *“SO does not rely on individual complexity to account for complex spatiotemporal features that emerge at the colony level, but rather assumes that interactions among simple individuals can produce highly structured collective behaviours.”* [25:188].

Within the ethology community, swarm intelligence is part of self-organisation (SO) and the two terms are sometimes used synonymously. Swarm intelligence can be considered the engineering implementation of SO.

Together the following four elements are the major mechanisms of SO [23]: 1) Interaction. An Individual should be able to make use of the results of its own activities as well as those of others. Trail networks can self-organise and be used collectively. 2) Positive feedback. These are simple rules that promote the creation of structures. Examples are recruitment and reinforcement in an ant colony. 3) Negative feedback. This is the opposite of positive feedback and helps to stabilise the collective pattern. In foraging, negative feedback is the limited numbers of available foragers or the weak pheromone trail. 4) Random fluctuations. Randomness is crucial since it enables the discovery of new solutions or food. An ant gets lost and discovers a new food source.

### Section 2.1.2 Communication

Communication is very important in swarm-based robotic systems. In Section 2.1 we looked at the mechanisms of self organisation (SO) and they require some form of communication. The nature and extent of communication between robotic swarm agents are vital to successfully achieve their goal. Pagello and Parker [26] distinguish between implicit and explicit communication and defines it as: *“implicit communication occurs as a side effect of*

*other actions, or ‘through the world’, whereas explicit communication is a specific act designed solely to convey information to other robots on the team”.*

Implicit communication is also known as stigmergy and is most often found in foraging and sorting tasks. Messages are sent by altering some aspect of the environment which is then sensed by another individual. The environmental changes can be intentional (trail-laying) or unintentional (sorting). These messages cannot be for specific individuals but is rather seen by any agent. This form of communication is very limiting.

### **Section 2.1.3 Ant behaviour**

A significant amount of swarm research is inspired by the ant. They are readily visible and respond well to laboratory testing. The ant is a classic model on which swarm intelligence is based. An individual ant is insignificant, weak and does not exhibit complex behaviour yet in a swarm they accomplish great things.

#### **Section 2.1.3.1 Foraging**

Cooperative foraging is one of the most important tasks in the study of multi robot cooperation. Drogoul and Ferber [28:1] note that foraging is “*widely accepted as the best illustration of ‘swarm intelligence’...*” and Cao, Fukunaga, and Kahng, [27:3,4]] describe foraging as “*one of the canonical test beds for cooperative robotics*”. Foraging is defined as the location and collection of objects and for insects these objects usually are food. In a robotic application the objects are task specific.

Cooperative foraging requires some form of communication. In the case of ants this communication is mostly indirect through changes in the world. Ants use trail-laying and trail-following behaviour when foraging. Each ant deposits a pheromone chemical when walking from the object (food) to its home and other foragers follows this pheromone trail. This process is called recruitment and when the trail-following decision is solely made on the presence of pheromones it is called mass recruitment (many ants follow the trail). Recruitment can be defined as communication that brings nest mates to some point in space where work is required. Thus recruitments can also be used for defence, nest building and a variety of other tasks.

### **Section 2.1.3.2 Learning**

The amount of foragers recruited affects the strength of the pheromone trail.

The pheromone chemical used decays over time and if the rate of laying the pheromone trail is slower than the chemical decay time, the trail will fade over time. This is seen as negative reinforcement. When foraging, the negative reinforcement might be due to inadequate numbers of available worker ants to utilise the newly found food (object) or there might be a closer source. The end result is that the negative reinforcement discourages other ants to follow the decaying trail as ants will almost always follow the strongest scent trail.

On the other hand if enough ants are recruited the trail scent gets stronger over time and becomes a pheromone highway that is difficult to ignore. This is positive reinforcement.

Most of the learning algorithms used in multi-robot systems are based on reinforcement.

### **Section 2.2 Layered Control System (Subsumption)**

In 1985 Rodney Brooks wrote an article, “A robust layered control system for a mobile robot” [2]. Brook’s paper described a layered control system which he named subsumption architecture. This paper played a major role in shaping the course of multi robot systems. Brook’s presented a novel control strategy that is both flexible and robust. He defines levels of competence for an autonomous mobile robot which functions as a specification for the desired behaviour over all environments it will encounter. For a wandering robot these could be the following:

0. Avoid contact with objects.
1. Wander around without colliding with anything.
2. Explore the world.
3. Build a map of the environment and plan routes.
4. Notice changes in the static environment.

Each level of competency includes a subset of the previous one. When designing the control system, a layer of control is designed for each level of competence starting at the lowest level, level 0. This level is tested and then the next one is developed. When the next layer is finished it forms with its predecessor the control layer for the current competence level. This means that for a level 1 competence controller, layer 0’s controller is also running which leads to the name layered control system. A top layer cannot function without the layers

below it (see Figure 1). Thus the control system is designed from the ground up (higher layers has higher capability and are developed later).

Each control level functions on its own and any interaction between layers happens through the world and not through the system. Brooks implemented each control level as an asynchronous finite state machine that is responsible for its own sensor inputs and actuator outputs. There is no centralise control or state and each controller does its task the best it can. Inputs to modules can be suppressed and outputs can be inhibited by higher level controllers. This is the mechanism by which higher level layers subsume the role of lower levels. A Layered control system has distributed representation and distributed computing.

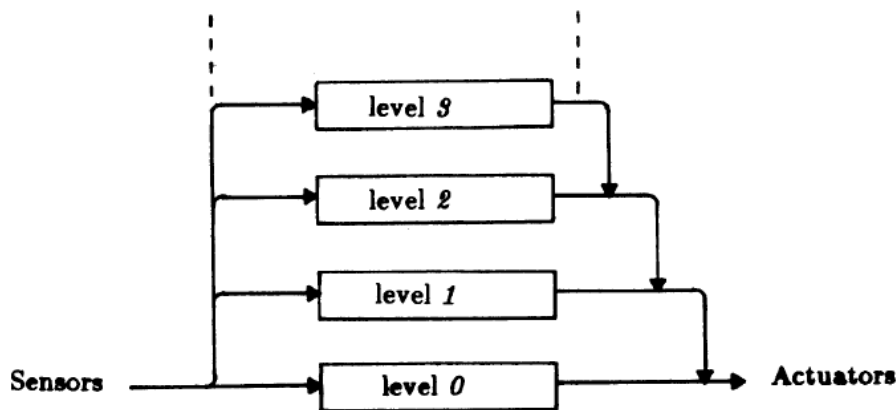


Figure 1. Layered control [2]

### Section 2.2.1 Basis/Basic Behaviour

Behaviour is defined as control laws that take advantage of the dynamics of a system to achieve a goal. Basis/basic behaviour forms a set of optimal minimum behaviours needed to achieve a goal. Maja Matarić describes basis behaviour as “*Basis behaviours are stable prototypical interactions between agents and the environment that evolve from the interaction dynamics and serve as a substrate for more complex interactions*” [10]. Matarić goes further and sets the criteria for basis behaviour selection as “*A basis behaviour set should contain only behaviours that are necessary in the sense that each either achieves or helps achieve a relevant goal that cannot be achieved with other behaviours in the set and cannot be reduced to them. Furthermore a basis behaviour set should be sufficient for accomplishing the goals in a given domain so no other basis behaviours are necessary. Finally, basis behaviours should be simple, local, stable, robust, and scalable*” [2:4].

Basis behaviours are intended as building blocks for higher-level goals.

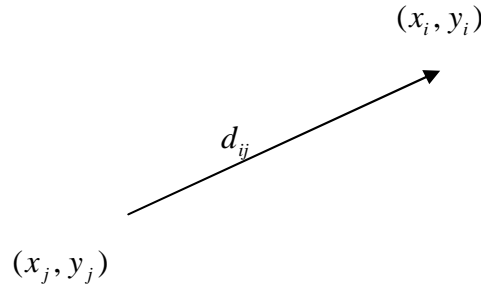
When considering the movement of an ant or a robotic agent, the following basic behaviour can be identified: safe wandering, following, dispersion, aggregation and homing. The following subsections describe each behaviour type in more detail.

The following conventions apply to the subsections below.

$\mathfrak{R}$  is the set of robots:  $\mathfrak{R} = \{R_i\} \quad 1 \leq i \leq n$

$p_i = (x_i, y_i)$  and  $p_j = (x_j, y_j)$  are 2-dimensional positional coordinates.

The travel distance between these two points:  $d_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$



This function returns all other robots within the neighbourhood:

$$N(i, \delta) = \{j \in i..n \mid d_{i,j} \leq \delta\}$$

for a given robot R with a distance threshold of  $\delta$ .

### Section 2.2.1.1 Safe-wandering

This is defined as the ability of a group of agents to move about while avoiding collision with obstacles and each other. Thus, agents should move around and maintain a minimum distance  $\delta_{avoid}$ .

$$\text{Instant velocity: } \left\| \frac{d \bar{p}_j}{dt} \right\| > 0 \text{ and } \forall(i) \quad d_{i,j} > \delta_{avoid}$$

Matarić devised the following behaviour algorithms [3]:

Velocity command:  $command\left(v\begin{pmatrix}\cos(\theta + \Delta) \\ \sin(\theta + \Delta)\end{pmatrix}\right)$

```
//Avoid Kin
Whenever an agent is within  $\delta_{avoid}$ 
    If the nearest agent is on the left
        turn right
    otherwise turn left

//Avoid Everything Else
Whenever an obstacle is within  $\delta_{avoid}$ 
    If an obstacle is on the right only, turn left.

    If an obstacle is on the left only, turn right.
    After 3 consecutive identical turns, backup and turn.

    If an obstacle is on both sides, stop and wait.
    If an obstacle persists on both sides,
        turn randomly and back up.

//Move Around
Otherwise move forward by  $\delta_{forward}$  , turn randomly.
```

**Figure 2. Algorithmic Pseudo Code for Safe-Wandering**

$\theta$  is the orientation of the robot R and  $\Delta$  is the robots incremental turning angle away from the obstacle.

### Section 2.2.1.2 Following

This is defined as the ability of an agent to move behind another retracing its path. Thus, the follower should maintain a minimum angle  $\theta$  between itself and the leader.

i = leader      j = follower

Projection of the followers speed with regards to the leader:



$$0 \leq \frac{d \bar{p}_j}{dt} \cdot \left( \bar{p}_i - \bar{p}_j \right)$$

$$\therefore 0 \leq \left\| \frac{d \bar{p}_j}{dt} \right\| \cdot \left\| \left( \bar{p}_i - \bar{p}_j \right) \right\| \cdot \cos \theta$$

But  $\theta$  must be as small as possible to ensure that the follower is accurately following the leader. When  $\theta$  is 0  $\cos \theta = 1$

$$\therefore 0 \leq \left\| \frac{d \bar{p}_j}{dt} \right\| \cdot \left\| \left( \bar{p}_i - \bar{p}_j \right) \right\|$$

Matarić devised the following behaviour algorithms [3]:

//Follow

Whenever an agent is within  $\delta_{follow}$

If an agent is on the right only, turn right.

If an agent is on the left only, turn left.

**Figure 3. Algorithmic Pseudo Code for Following**

Velocity command:  $command(v_0 \cdot \hat{p})$

$$\therefore command \left( v_0 \cdot \frac{\bar{p}_{leader} - \bar{p}_{follower}}{\left\| \bar{p}_{leader} - \bar{p}_{follower} \right\|} \right)$$

### Section 2.2.1.3 Dispersion

This is the ability of a group of agents to spread out in order to establish and maintain a minimum inter-agent distance ( $\delta_{dispersion}$ ).

$$\forall(j) \quad d_{i,j} > \delta_{dispersion} \quad \text{But } \delta_{dispersion} > \delta_{avoid}$$

Dispersion can be seen as an extension of safe-wandering.

Matarić devised the following behaviour algorithms [3]:

*//Disperse*

Whenever one or more agents are within  $\delta_{dispersion}$   
move away from Centroid\_disperse.

**Figure 4. Algorithmic Pseudo Code for Dispersion**

$$\text{Velocity command: } command \left( -v_0 \cdot \frac{C(i, \delta_{disperse}) - \bar{p}_i}{\|C(i, \delta_{disperse}) - \bar{p}_i\|} \right)$$

Dispersion should be seen as an ongoing task in which the agents maintain a specified distance to other agents. It is important to measure the distance accurate to ensure a proficient dispersion algorithm.

#### **Section 2.2.1.4 Aggregation**

This is the ability of a group of agents to gather in order to establish and maintain a maximum inter-agent distance ( $\delta_{aggregate}$ ).

$$\forall(j) \quad d_{i,j} < \delta_{aggregate}$$

Aggregation is the inverse of dispersion.

Matarić devised the following behaviour algorithms [3]:

*//Aggregate*

Whenever nearest agent is outside  $\delta_{aggregate}$   
turn toward the local Centroid\_aggregate, go.  
Otherwise, stop.

**Figure 5. Algorithmic Pseudo Code for Aggregation**

$$\text{Velocity command: } command \left( v_0 \cdot \frac{C(i, \delta_{aggregate}) - \bar{p}_i}{\|C(i, \delta_{aggregate}) - \bar{p}_i\|} \right)$$

### Section 2.2.1.5 Homing

This is the ability to find a particular region or location. The agent must decrease the distance between itself and home.

$$\forall(j) \quad \frac{d \bar{p}_j}{dt} \bullet \left( \bar{p}_j - p_{home} \right) < 0$$

Matarić devised the following behaviour algorithms [3]:

```
//Home
Whenever at home
    stop
    otherwise turn toward home, go.
```

Figure 6. Algorithmic Pseudo Code for Homing

Velocity command:  $command \left( v_0 \cdot \frac{\bar{p}_{home} - \bar{p}_i}{\|\bar{p}_{home} - \bar{p}_i\|} \right)$

### Section 2.2.2 Higher level behaviour. Combining Basic Behaviour

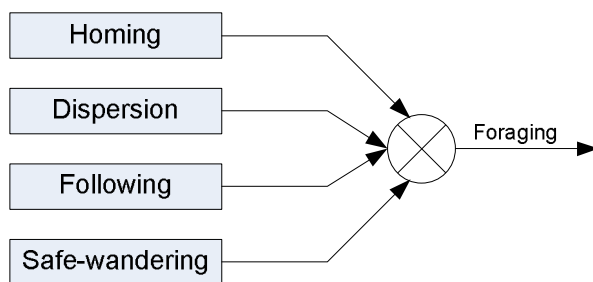
Basic behaviour forms the lowest level of control. Higher level behaviour is formed by a subset of basic behaviours. The challenge is how to combine these basic behaviours. One possible combination is to choose mutually exclusive basic behaviour and combine them. Thus there is no arbitration of control outputs. Care should be taken not to use state to choose between mutually exclusive behaviours, but rather use the environment (the world) as cues for behaviour selection. Mutual exclusive behaviour is sufficient for arbitration in systems that perform a basic behaviour at a time but for more complex systems a different arbitration technique is required.

In complex systems basic behaviour can be combined into one control output. Thus each individual behaviour output must be weighted and combined into one control output. Usually the output of each basic behaviour controller is in the form of a direction or velocity vector, so the weighted sum of these vectors will produce the higher level behaviour control output.

## Section 2.2.3 Typical Higher level Behaviours

### Section 2.2.3.1 Foraging

The goal is to collect items from the environment and bring them to a common location, home. While foraging, it is important for an agent to be able to determine other agent's state. This is to ensure that the agent that collected an item will not follow empty handed agents. Thus there are two states: an agent with an item on its way back home and an agent without an item busy looking for items. The state determination will typically be done through explicit communication.



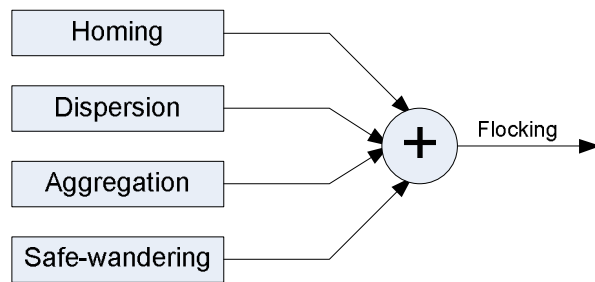
**Figure 7. Basic behaviour arbitration for foraging**

Figure 7 shows the behaviour for foraging. When this task is started the agent will make use of dispersion and safe-wandering. Safe-wandering is used to avoid collisions while dispersion assures that a bigger area is covered in the search. When the agent finds an item, homing is triggered. An agent can also see that the homing was triggered by an external or worldly event, namely, the presence of the item. When the agent reaches home and delivers the item, dispersion and safe wandering is triggered (once again by a world condition = no item). It is also possible for a single agent to do foraging.

Following is triggered when the agent is on its way to home and it encounters another agent which has an item (worldly event).

### Section 2.2.3.2 Flocking

Flocking is the selective motion of individuals in which all agents within sensing range stay within the flocking range of their neighbouring agents. All agents move towards a common destination usually referred to as home. This goal distinguishes flocking from aggregation.



**Figure 8. Basic behaviour arbitration for flocking**

Aggregation keeps the agents from going too far from each other while dispersion keeps them from getting too close to each other. Safe-wandering prevents collisions and homing ensures that they all go to the common destination. Thus it is evident that the output of the controller must be a weighted sum of each behaviour (see Figure 8).

### ***Section 2.3 Swarm implementation of the Ant's Basic Behaviour***

Mark Russel Edelen [1] implemented the ant basic behaviour and did some simulations and experiments. In his experiments he used no implicit communication. The basic agent was built from a Lego Mindstorm differential drive robot with evaporative ink used for the pheromone trail. He focused on foraging and developed a system of robots capable of utilising trail-laying and following techniques.

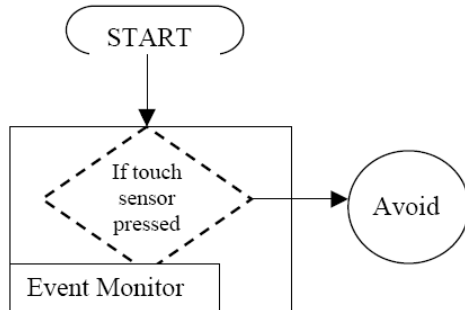
Edelen implemented the following basic behaviours:

- Avoid – Collision detection and response.
- Wander – Searching randomly for targets.
- Follow – Following an ink trail to the food source.
- Homing – Picking up a target, moving to home, and dropping the target.

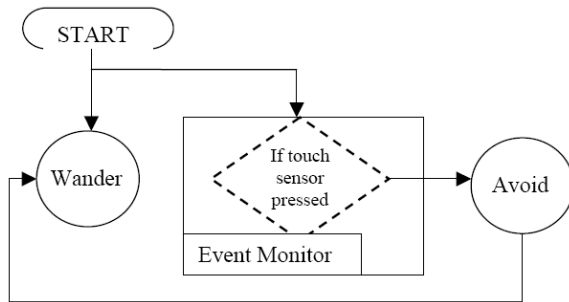
Arbitration between the behaviours is based on sensory inputs. Three levels of competence are designed within this control architecture, Levels 0, 1, and 2 with level 0 being the lowest level.

- Level 0: Avoid.
- Level 1: Wander.
- Level 2: Follow and Homing.

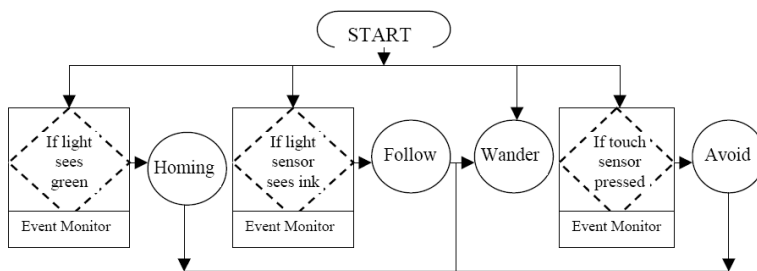
Figure 9, Figure 10 and Figure 11 shows a graphical representation of the competency levels. Observe the inclusion of the lower levels into the higher levels and the arbitration through sensor inputs.



**Figure 9. Competence Level 0 [1:184]**



**Figure 10. Competence Level 1 [1:184]**



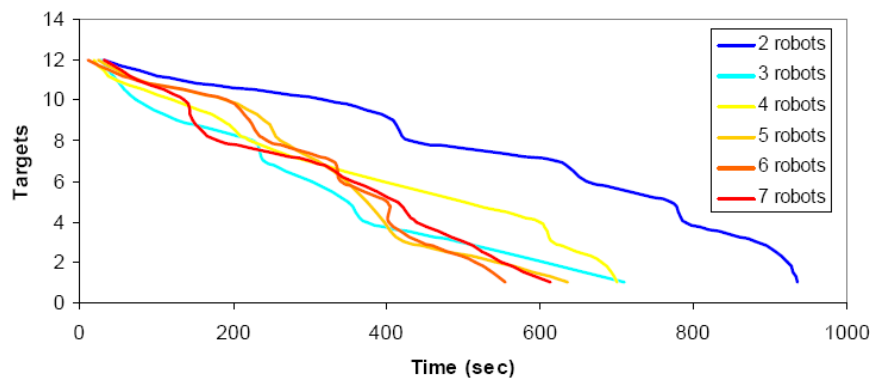
**Figure 11. Competence Level 2 [1:185]**

The avoid behaviour is responsible for collision detection and avoidance and forms a reactive controller. Wandering is used to search the foraging field for targets (“food”). Follow instructs the robot to follow the ink (pheromone) trail after recruiting. Once a target is collected the homing behaviour is used to go back to a centralised position (“home”).

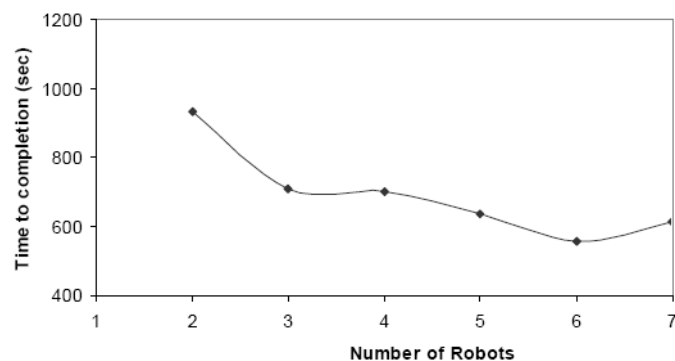
### Section 2.3.1 Experiment

In the experimental setup Edelen used 1-7 robots. Washers were used as targets (“food”) and the robots could determine if other agents has already picked up a target. This information was used to determine if homing action should be taken or following (an agent with a target is assumed to be on its way to home). The evaporative ink pheromone trail introduced the positive and negative reinforcement.

In one of the experiments Edelen introduced 16 targets at a single location inside the foraging field. After the first discovery of the targets a trail (ink) is laid back to home, and through positive feedback this trail is reinforced and more robots discover the trial. A well-established ink trial emerges as a result of stigmergy and positive feedback. The experiment was conducted with 2 to 7 robots. Figure 12 and Figure 13 shows a steadily decrease in collection time as the number of robots is increased.



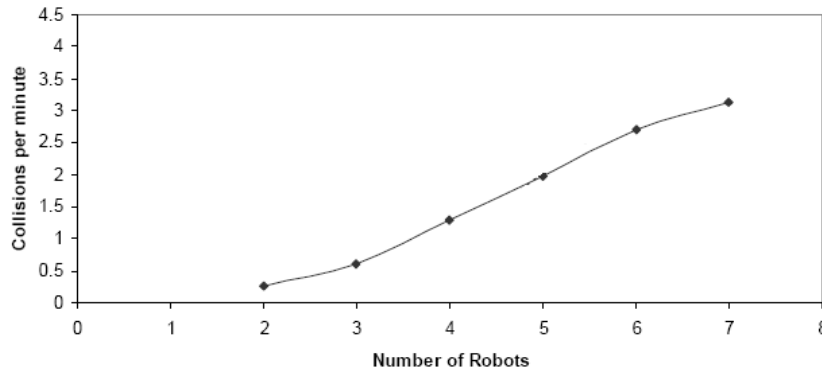
**Figure 12. Remaining targets vs. Time for single cluster of targets (1 cluster) [1:205]**



**Figure 13. Collection time versus number of robots (1 cluster) [1:206]**

As the number of robots is increased collisions occurs more frequently (see Figure 14). Cooperation slows down as the collisions increases. This would suggest that for a given

situation, there are an optimal number of agents, and increasing the agent count pass this value has a negative impact on task completion time. The detrimental effect of collisions between robots negated the beneficial effect of cooperation.



**Figure 14. Collision frequency versus number of robots (1 cluster) [1:207]**

The opposite is also true and with small robotic groups (3 and smaller) it was not possible to lay a strong reinforced trail.

### **Section 2.3.2 Weaknesses**

The experiment shows the benefit of swarm cooperation but the results were marginal due to the low agent count. In nature swarm intelligence depends heavily on agent numbers and it is not possible to fully demonstrate the potential with 7 agents.

Due to the construction and sensor limitations of the robots, the robots showed a significant variation between trails. Trails were not followed with high accuracy. Sometimes some of the targets were passed without detecting them.

Another limitation is the agent battery life. Approximately 20 minutes of continuous operation was possible with a new set of alkaline batteries. During the 20 minutes the robot speed also varied as the battery voltage changed. This introduced a time limit on the experiments and it was not always possible to demonstrate the convergence. The drift in agent speed also introduced errors. The design did not support rechargeable batteries and the author used more than 200 batteries [1:191].



## Chapter 3 : Reconfigurable Computing

Electronic design has become very expensive especially with the invention of the BGA package. These devices have high ball count placed at a small pitch. Some of Xilinx's devices for example contain 668 balls within a 17mm x 17mm area. It is very difficult, if not impossible, to place these devices at the correct location by hand. To further complicate matters these devices must be soldered inside a reflow oven programmed at the correct temperature profile to ensure good quality solder joints and optimal component life. Once all of this is completed it is still not certain if each ball in the BGA was soldered, without creating a short or a dry joint, correctly. To confirm proper reflow each BGA is X-rayed. All of this increases the board manufacturing cost. The only way to overcome this cost is to avoid going through multiple design iterations and try to consolidate all requirements in one generic reconfigurable hardware platform. In this philosophy the FPGA forms an impressive building block allowing the designer a degree of freedom at design time.

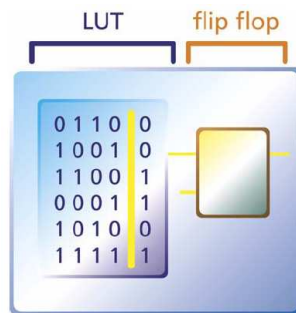
The swarm intelligent robot platform put forward in this design is intended as a research tool. It is impossible to speculate over the implementation detail and system configuration. For this reason reconfigurable computing forms an integral part of this solution thus enabling research activities of different size and complexities.

### **Section 3.1 *FPGA***

In 1984 Ross Freeman designed the first FPGA. Unlike the PLD's from that time, the FPGA focuses on flexibility through the use of programmable interconnections. At the time transistor cost was high and the first FGPA's where modest in size. Ross postulated that transistors, because of Moore's Law (the doubling of transistor density every 2 years), would be getting less expensive and therefore less precious every year. This proved to be a novel idea and today the FPGA has grown aggressively in market share and density.

All FPGA devices have a similar structure comprising of an array of logic blocks surrounded by routing lines connected through switch boxes. The routing lines can consist of different routing primitives. These primitives can span across single and multiple logical blocks. The selection of routing primitives has a profound effect on the signal speed and power consumption.

The content of the logical block or cell varies between different FPGA's and vendors, but are generally based on a logical element consisting of a configurable combinatorial logic element and a register (see Figure 15). The configurable combinatorial logic element is usually implemented as a n-input look up table (LUT) and the register as a flip-flop. The LUT determines the output based on the inputs.



**Figure 15. Logic Cell**

Real world FPGA's are more sophisticated than the model describe above. Logic blocks include hardware for special functions such as carry chain logic and multiplexers. The LUT can also be used for storage, frequently referred to as distributed RAM. Input and output buffers can be configured to support different voltage standards and speeds.

The floor plan of the FPGA does not conform to a homogeneous structure. To increase the overall performance of the device, different sections within the die contains fixed hardware blocks such as multipliers, RAM, DSP blocks, clock generator circuits and even complete microprocessors.

Configuring the FPGA involves connecting logic blocks as well as other fixed blocks and setting up the switch boxes. The configuration bits controlling the interconnection, logic and storage is collectively referred to as the configuration of the FPGA. The configuration bits are stored in SRAM inside the FPGA. Other technologies exist which are flash based, but SRAM devices are of more importance due to their ability to reconfigure an unlimited number of times. The reconfiguration speed of SRAM FPGA's is also fast in comparison to other technologies. Each time the device is powered the configuration bits must be transferred from the external storage to the SRAM inside the FPGA.

## **Section 3.2 Dynamic Reconfiguration**

Dynamic reconfiguration is the ability to reconfigure the FPGA during runtime. The FPGA exposes its configuration interface through a JTAG port. Usually a manufacture specific JTAG pod is used to erase, configure or test the device through the JTAG port. An external microprocessor can take advantage of this interface and use it to download new bit streams to the FPGA at runtime.

This introduces greater system functionality and flexibility that allows the designer to use a smaller FPGA when possible. There are however certain conditions and limitations when doing dynamic reconfiguration. It is not possible to reconfigure only portions of the FPGA and therefore the whole FPGA is erased and reprogrammed. The time associated with the reconfiguration process is directly related to the size of the FPGA and large device can take some time especially when reconfiguring it through the serial JTAG interface.

During the reconfiguration process the FPGA logic is held in reset state. This makes it impossible for the FPGA to reconfigure itself and an external device is required to fulfil this task. Usually an embedded microcontroller is used, but this increases the PCB complexity, system cost and power consumption. A more efficient approach would be for the FPGA to reconfigure portions of its logic without affecting the rest or putting them in reset. This would make the FPGA the ideal reconfigurable system on chip.

### **Section 3.2.1 Dynamic Partial Reconfiguration**

Dynamic partial reconfiguration addresses some of the limitations of dynamic reconfiguration such as long configuration time. Devices that fall in this category can reconfigure sections of the FPGA logic without affecting the bit stream data of the rest. It is possible to divide the FPGA into reconfigurable sections, but the sections are usually large and do not lend itself to fine granularity. The other major limitation is that all logic is held in a reset state while partial reconfiguration takes place. This makes self reconfiguration impossible.

### **Section 3.2.2 Dynamic Partial Self Reconfiguration**

Dynamic partial self reconfiguration (DPSR) addresses all the limitations discussed in the previous sections. These devices can dynamically reconfigure a section of the FPGA logic without changing the bit stream data or state of the unaffected logic. This implies that the

unaffected logic can continue to function while partial reconfiguration is taking place. The FPGA can be reconfigured by an external JTAG interface or a duplicate internal reconfiguration interface. *The combination of these advantages makes it possible to remove the external embedded microcontroller and replace it with a soft processor core inside the FPGA.* The processor core will always run on a fixed section in the FPGA that can never change dynamically. When applying this technology, the FPGA is divided into at least one fixed area containing the soft core and multiple reconfigurable areas under the control of the soft processor core. It should be noted that there are vendor specific rules that governs the division of the FPGA into sections and it is not possible to place a reconfigurable section anywhere in the FPGA.

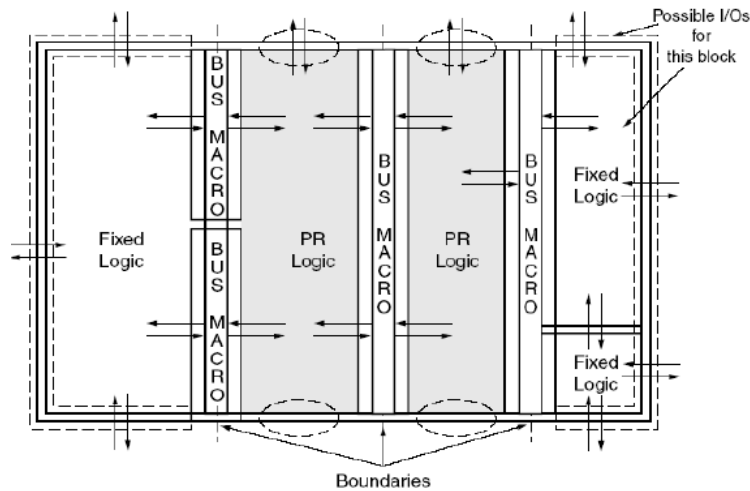
### **Section 3.3 Xilinx architecture**

Xilinx offers three FPGA families known as the Spartan3, Virtex4 and Virtex5 devices. The Virtex5 device is their latest offering but due to its high cost will not be considered in this research. Both the Virtex4 and Spartan3 support dynamic partial reconfigurations. There are fundamental differences in the functionality of each of these families and certain rules and limitations apply.

#### **Section 3.3.1 Dynamic Partial Reconfiguration**

The Spartan3 family is marketed as a low cost FPGA and is the most restrictive when used for partial reconfiguration. When laying out the reconfigurable modules, care must be taken to ensure that the height of the module is always the full height of the chip. The width of the module should also be at least 4 slices or more. All modules should always be placed on 4-slice boundaries. All logic resources within the module boundary forms part of the reconfigurable area, this includes IOB, TBUF, RAM, multiplier and routing resources. All IOB's immediately above the top and below the bottom of the module are also included in the reconfigurable area. If a module occupies the left or right most area of the FPGA all the IOB's on that edge is also included in the reconfigurable area. Figure 16 shows a typical reconfiguration layout for a Spartan3 device that adheres to the above rules. The major disadvantage of the Spartan3 is the reset state in which all unaffected modules are held while a module is reconfigured. This renders the rest of the FPGA useless during a module reconfiguration and is termed glitched reconfiguration. It is however possible to design support into the modules for this phenomena, but it introduces an additional level of

complexity. Glitched reconfigurable architecture can never be used when self partial reconfiguration is required.



**Figure 16. Typical reconfigurable logic for the Spartan3**

When concerned with dynamic partial reconfiguration, the biggest difference between the Virtex4 and the Spartan3 is the introduction of clock regions and the redefinition of the reconfigurable module boundaries. The minimum module height is the height of a clock region. Figure 17 shows the clock regions (bold blue lines) for the Virtex 4LX15. The Virtex 4 introduces a new reconfiguration granularity. In the past a reconfigurable module was always the height of the FPGA device. The Virtex 4 introduces a reconfigurable tile (frame height = 1 column = 16 CLB's or slices).



**Figure 17. Virtex4 LX15 Floor plan (rotated by 90 deg)**

The LX15 contains 8 clock regions, 4 on the left side and 4 on the right side. The height of a clock region is 32 IOB's (16 CLB's) and the width is half the die width. All logic regions in the middle of the chip die (IOB's, DCM's) are included in the left clock regions. In Figure 17 the purple blocks are block RAM (BRAM) and the light blue blocks are DSP48 blocks. Table 1 shows the location of the clock regions and the IO ports included in a region. Because all

IOB's in a configuration area forms part of the configuration, it is important to know which IO banks are included when working in a clock region.

Clock region	Name	Row	Column	IO banks included
1	X0Y0	0	0	4 & 7
2	X0Y1	1	0	2 & 7
3	X0Y2	2	0	1 & 5
4	X0Y3	3	0	3 & 5
5	X1Y0	0	1	8
6	X1Y1	1	1	8
7	X1Y2	2	1	6
8	X1Y3	3	1	6

**Table 1. Virtex4 LX15 clock region definition**

The clock logic is always separate from the reconfiguration logic and is not affected by a reconfiguration. When designing the fixed configuration modules all reconfigurable areas should be specified and the clock logic defined. The clock logic cannot be changed dynamically.

The Spartan3 and Virtex4 devices can be configure via their JTAG ports which is an external serial interface. Large devices can take a long time to configure when transferring the bit-stream serially. To overcome this delay in configuration, Xilinx introduced the SelectMAP port which is a parallel external configuration interface. Both families support this interface and an external microprocessor can be used to configure the device through this interface.

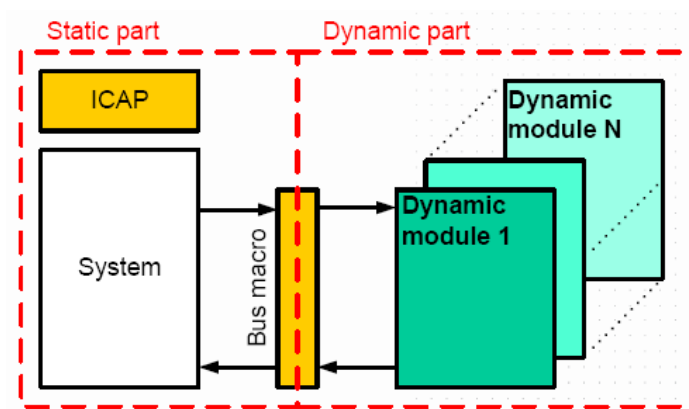
### **Section 3.3.2 Dynamic Partial Self Reconfiguration**

From Section 3.2.2 it is evident that the Spartan3 cannot do DPSR. The only device from Xilinx that supports DPSR is the Virtex 4 & 5 families. None of the other major manufacturers (Altera, Actell and Lattice) supports this technology and Xilinx has positioned themselves as the only company to offer glitchless dynamic partial self reconfiguration.

The Virtex family uses the Configuration Access Port (ICAP) for reconfiguration. The ICAP allows access to configuration data in the same manner as SelectMAP. ICAP has the same interface signalling as SelectMAP, other than the data bus, which is separated into read and write data buses. ICAP has a chip select signal (CS), a read-write control signal (RD), a clock

(CLK), a write data bus (DIN), and a read data bus (OUT). ICAP can be configured to two different data bus widths, 8 bits or 32 bits. The ICAP interface can be used for read back and partial reconfiguration but cannot be used to do a full configuration. With no handshaking mechanism ICAP can be clocked up to the maximum frequency of 100MHz. There are two ICAP sites on the Virtex 4, one at the top and one at the bottom of the die. The top site is the default port at start-up where after one can switch to the bottom one.

Figure 18 shows a typical structure of a self reconfigurable system. The system block contains the static soft processor core which will do the self partial reconfiguration through the ICAP port. The dynamic modules are depicted as small blocks to illustrate the smaller size compared to the height of the device.

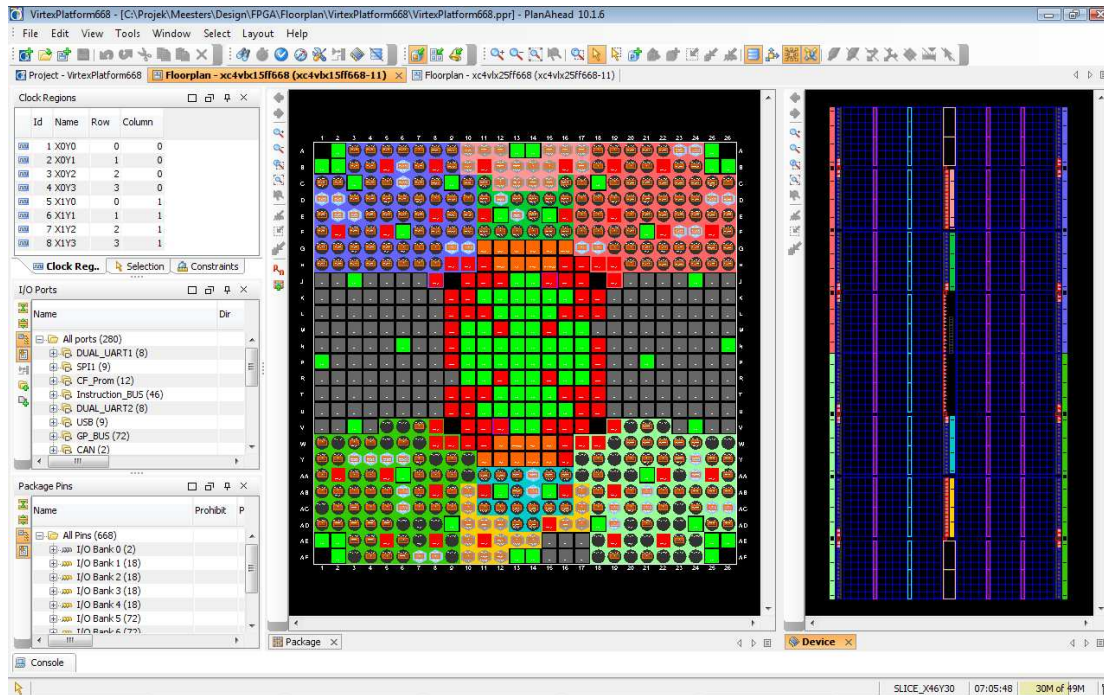


**Figure 18. Self Reconfiguration block diagram**

### Section 3.3.3 Tools

Partial reconfiguration is an advanced topic in FPGA design and requires a lot of effort to design. One of the major difficulties is finding the optimal floor plan for the design. Because of this Xilinx has developed PlanAhead. PlanAhead is a GUI based tool to assist the designer in floor planning the design, to ensure that all the design rules and criteria are met. The PlanAhead software is only available through a special access program for which one must apply. Xilinx philosophy is that this topic requires a lot of advanced support and they cannot allow the general public to access it. It should be noted that it is possible to implement partial reconfiguration without the PlanAhead tool, but this is significantly more complicated and increases the design time.

Figure 19 shows the screen layout of PlanAhead when used to do floor planning. The right side shows the die of the chip. The top view and pin locations is shown in the centre window. Pin description and type is also shown and banks are indicated through different pin colours. The left side shows the clock regions, package pins and the design I/O ports. In the design, signals can be grouped under a common descriptive name.



**Figure 19. Xilinx PlanAhead software**

After the signal description, types and the location have been entered, it is possible to do some simulations and checks. Of these the most important ones are the signal type compatibility test per FPGA bank and the Weighted Average Simultaneous Switching Output (WASSO) simulation used to estimate ground bounce in the FPGA. In the signal type compatibility test PlanAhead looks if the logic types are compatible within a bank. Each bank has its own I/O voltage supply pins and the logic types within the bank must be compatible with the I/O voltage.

The WASSO analysis takes into account the driver type, slew rate and drive strength of each pin. WASSO is computed first on a per I/O bank basis, then it is calculated for two adjacent banks, and finally calculated across all banks to determine the effective WASSO for the entire package. The results are compared to the WASSO limits published by Xilinx. The



intent of the WASSO analyses is to limit the ground bounce immediately at the output of the FPGA.

The ISE Design suite does not support partial reconfiguration. Xilinx offers a Partial Reconfiguration Tools overlay from their Partial Reconfiguration Early Access Lounge which must be installed over an existing ISE installation. The overlay will enable ISE to support partial reconfiguration design. Xilinx also recommend that this setup is solely used for partial reconfiguration design. For all other projects a separate ISE installation must be used.

### ***Section 3.4 FPGA Power considerations and reduction techniques***

The subsections below look at various techniques to lower the power consumption of a FPGA. Practical experiments are looked at to quantify the power saving potential.

#### **Section 3.4.1 Static and dynamic consumption**

The power consumption in the FPGA has two major components, static power consumption and dynamic power consumption. The static consumption is due to gate oxide tunnelling current, subthreshold conduction of MOS transistors and leakage in the reversed bias junction. As the chip die geometry becomes smaller, the leakage current increases significantly and becomes a major component of the power consumption. With a small and slow design such as this, the static consumption contributes to almost half of the total device consumption. In this research controlling the temperature of the FPGA core will be used to limit the static consumption.

The main source of dynamic power consumption is due to the charging and discharging of capacitances in the integrated circuit. The dynamic power consumption can be modelled by the following formula:

$$P = \sum (C \cdot V^2 \cdot f)$$

Where C is the parasitic capacitance of each part of the circuit, V is the supply voltage and f is the switching frequency of the circuit. Since there is a quadratic relationship between the voltage and the dynamic power consumption, reducing the FPGA bank voltage will reduce the dynamic power significantly. Managing the clock frequency will also lower the dynamic power consumption ( see Section 3.4.2 ).

### **Section 3.4.2 Clock Frequency**

The soft-cores and peripherals considered in this study are synchronous. There is a master clock driving the soft-core and from this clock a peripheral clock is generated. These clocks are used in all the synchronous circuits.

All logic gates and electronic components have both an input and an output capacitance. The PCB traces connecting the different components also contain a capacitive component. These capacitance values add up, and must be charged on every low to high transition and discharge on every high to low transition. The charging and discharging of this parasitic capacitance consumes power and increases the total device consumption. The only effective way to limit the power consumption is to scale the system clock frequency.

### **Section 3.4.3 Communication primitives**

Routing is one of the most complex tasks of FPGA design and this reflects in the abundance of interconnect types available: There are CLB-internal lines, direct lines, double lines, hex lines, long lines (see Figure 20 & Figure 21), each possessing individual electrical capacitance and skew characteristics. Fortunately, there is software which will perform automatic placement and routing of logic on the FPGA.

The Xilinx ISE design tool has multiple optimisation algorithms. The performance and optimisation algorithms are applied first. These algorithms are setup by the designer to favour one or the other or to apply a balanced approach. After applying these algorithms, if enabled, ISE will also perform a power reduction algorithm. This ensures that the lowest power consumption is achieved without severely impacting the size or performance of the design. The power save algorithm takes into consideration the different routing primitives and the power reduction potential. See Section 4.1.5 for implementation detail and results.

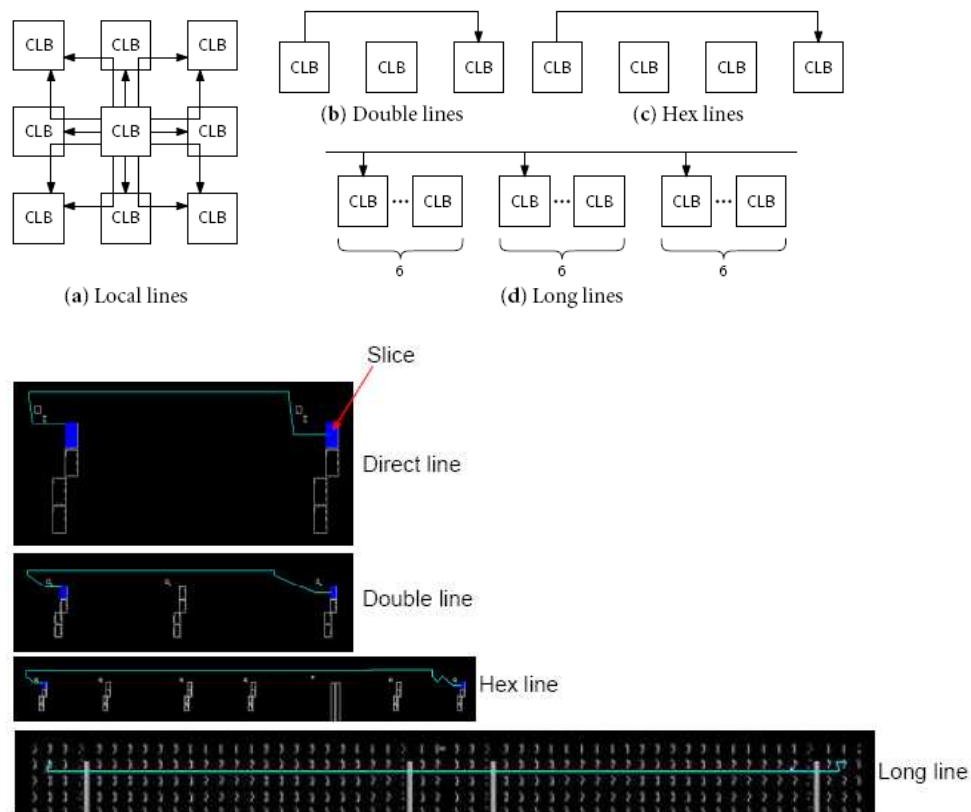


Figure 20. Hierarchical interconnect resources [13:176]

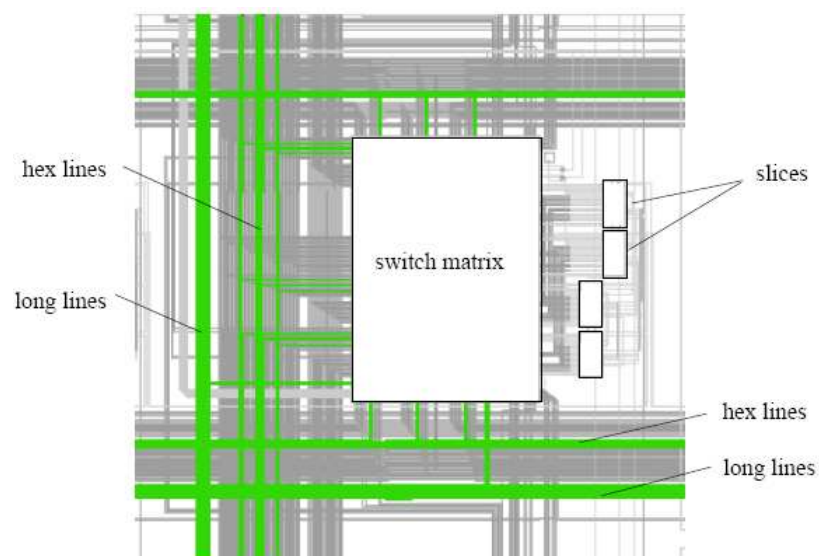
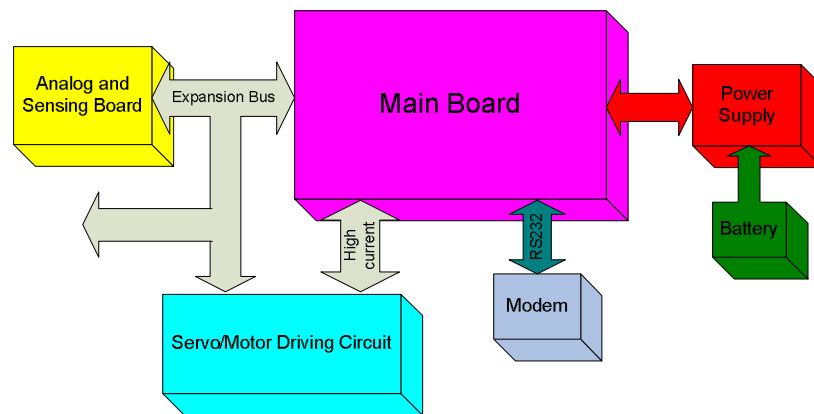


Figure 21. One Virtex 4 CLB showing routing lines

## Chapter 4 Design

Chapter 2 and Chapter 3 discussed the literature study that forms the basis of the hardware design. It is important to understand the underlying concepts before embarking on the design. The developed platform should sustain current and future research areas as outlined in Section 1.1 and must be applicable to this field.

This chapter will focus on the hardware design, which consists of component selection, HDL design, hardware design, board layout and software design.



**Figure 22. System block diagram**

Figure 22 shows the block diagram of the system. The main board functions as the brain of the system and is responsible for processing sensor inputs and generating control outputs. At the heart of the main board lies a dynamic self-reconfigurable FPGA.

The power supply is deliberately moved off the main board due to cost. The cost per square centimetre of the main board is much higher than the two layer power supply board.

The motor driving and analogue circuits are combined on a single board referred to as the peripheral board. Through this board the main board can acquire sensor inputs and send control instructions to the actuators. The motor driving circuit supports stepper and DC motors. Communication between the main board and the peripheral board takes place through the I2C, SPI bus and discrete I/O lines.

To facilitate higher behaviour within the swarm a RF modem is include on the peripheral board. The modem is optional and makes use of the Zig-Bee low power technology.

For autonomous behaviour the robotic agent must be able to navigate accurately. The peripheral board contains a GPS receiver with high sensitivity for indoor use.

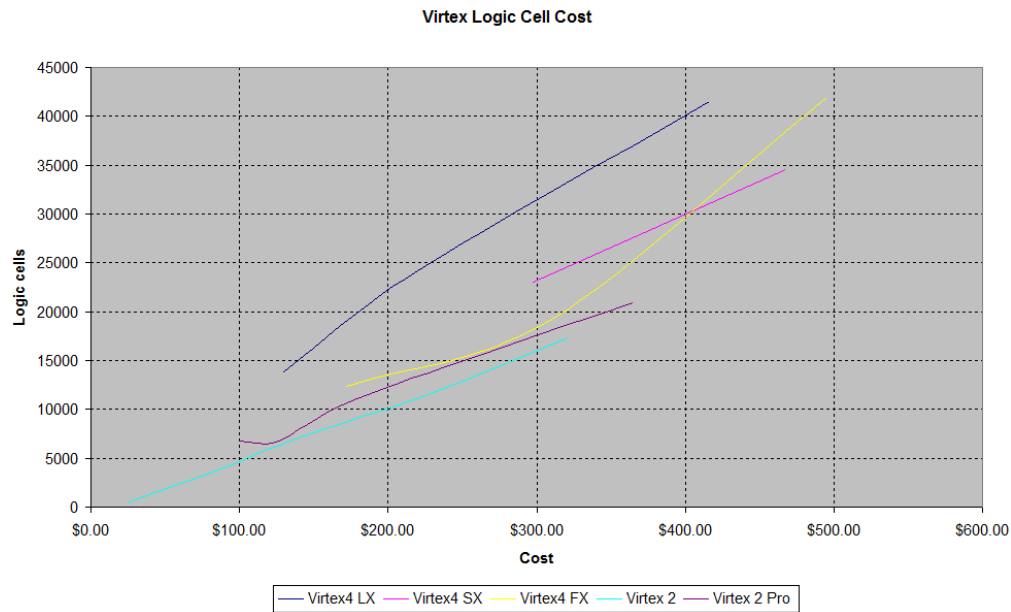
## **Section 4.1 Main Board**

This section explains the design philosophy behind the development of the main CPU board which forms the heart of ASRA. Component selection, soft CPU, peripheral design and hardware design is discussed.

### **Section 4.1.1 FPGA selection**

There are only two vendors on the market that supports partial self reconfiguration: Xilinx and Atmel. Of these two only Xilinx actively advertise and support the technology and have a complete tool chain to simplify the design phase and produce reliable results. Xilinx also gives the partial reconfiguration (PR) designer access to the PR lounge which functions as a portal to PR information, documentation, technical support and tool updates. When taking all of this into consideration, it is clear that Xilinx is the better option.

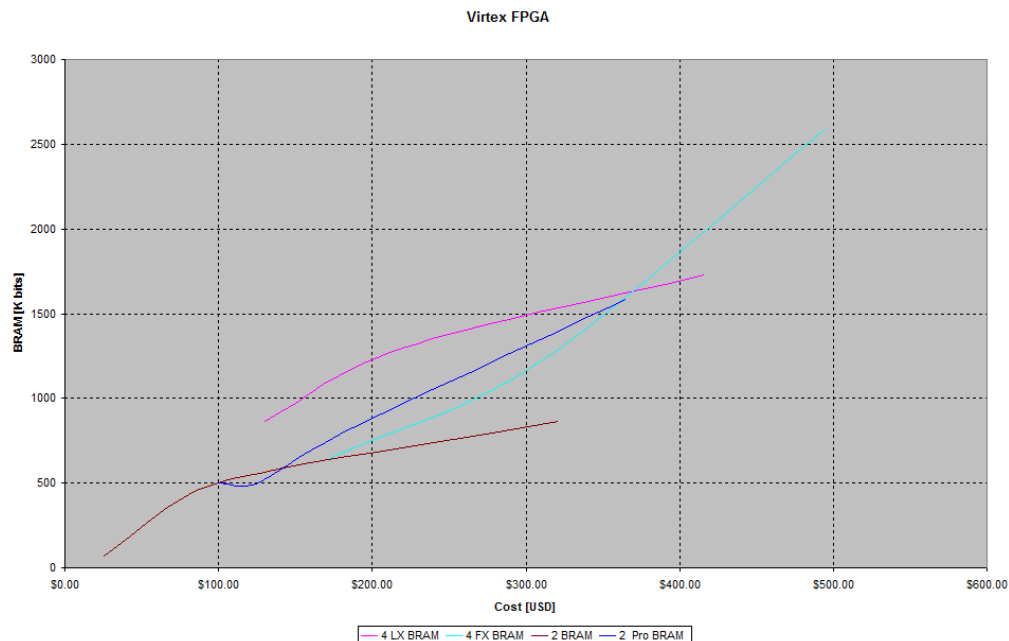
PR is supported by the Spartan, Virtex 4 and Virtex 5 family. The Spartan 3 does not support glitch-less PR which is a requirement for self PR, and the Virtex 5 is too expensive. This leaves the Virtex 4 that contains a SX, LX and FX derivative. The FX contains an IBM Power PC processor and the SX is optimised for DSP applications. The LX has the highest density and is optimised for speed.



**Figure 23. Virtex FPGA cell cost ratio**

Figure 23 shows the cost implication of the different Virtex devices including the older Virtex2 devices. When considering the cell to cost ratio it is clear that the Virtex4LX device is the best choice.

Figure 24 shows the BRAM to cost relationship for the Virtex family. BRAM plays an important role when implementing a soft processor core, especially if the application binary is loaded in the configuration PROM. Once again, the Virtex 4LX is the best option.



**Figure 24. Virtex FPGA BRAM cost ratio**

In light of Figure 23, Figure 24 and the fact that Xilinx graciously sponsored all the tools required for this project, it was decided to use the Xilinx Virtex4LX15 FPGA. It should be noted here that HDL implementation tools are very complex and manufacturer specific and for this reason no open source tools was considered.

### **Section 4.1.2 Processor Soft Core**

The FPGA's size has grown to a point where it is possible to implement single and multiple processors inside. These soft processor cores enables the designer to change or enhance the processor architecture where needed. The designer is also in control of the technology, he can make changes to the processor architecture and include or exclude certain peripherals. There is also no manufacturer involved that can decide to stop production of the processor.

There are numerous soft processor cores in the open source arena. Most of these are not highly optimised. Usually the level of optimisation is evident in the slice count used by the soft core.

Table 2 shows all the soft processor cores considered for the design. All the cores use the GCC tool-chain. The size of the core was used as an optimised and power consumption metrics. The Manik core is a highly optimised core. The core was synthesised in the Virtex LX15 device after which a small C project was compiled using the GCC compiler for this processor. After a few experiments it became apparent that the GCC compiler port contained bugs. This core is not maintained anymore and was not chosen.

The plasma core was synthesized successfully but cannot do unaligned memory access. This limitation is due to a MIPS patent on unaligned memory access. It is possible to use this core in a design, but it is rather decided to go for a fully functional architecture.

Name	Pipes	Lang.	BUS	Cache	IP Interface	Slices	4-inputs LUT's	Compiler
ASPIDA DLX	5	Verilog	Harvard.	None	Wishbone Chain	1585	2815	gcc
LEON3 Sparc V8	7	VHDL	Harvard.	1 - 256K instruction & data	AMBA	2979	5450	gcc
Plasma MIPS 1	2 or 3	VHDL				2021	3592	gcc
Aquarius SuperH-2	5	Verilog	Common code, data and peripheral bus	None	Wishbone	3072	5734	gcc
aeMB DLX	3	Verilog	Harvard.	None	Wishbone	1116	2009	gcc
Openrisc 1200 RISC	5	Verilog	Harvard.	0 - 8KB instruction & data	Wishbone	3100		gcc
Manik RISC	4	VHDL (no source)	Data & instruction same addr space	yes	Wishbone	809	2992	gcc
Lattice Mico32 RISC	6	Verilog	Harvard.	0 - 8KB instruction & data	Wishbone	2100	1830 to 2230	gcc

**Table 2. Soft Cores review**

Aspida is an asynchronous design that is very complex. Asynchronous design techniques were used to reduce the power consumption of the core. The Xilinx power calculation tool estimates its total power consumption at 451 mW. Aquarius and aeMB were also synthesized and their power consumption was estimated 430 mW and 425 mW. Thus it is evident that the additional complexity did not provide an adequate power reduction.

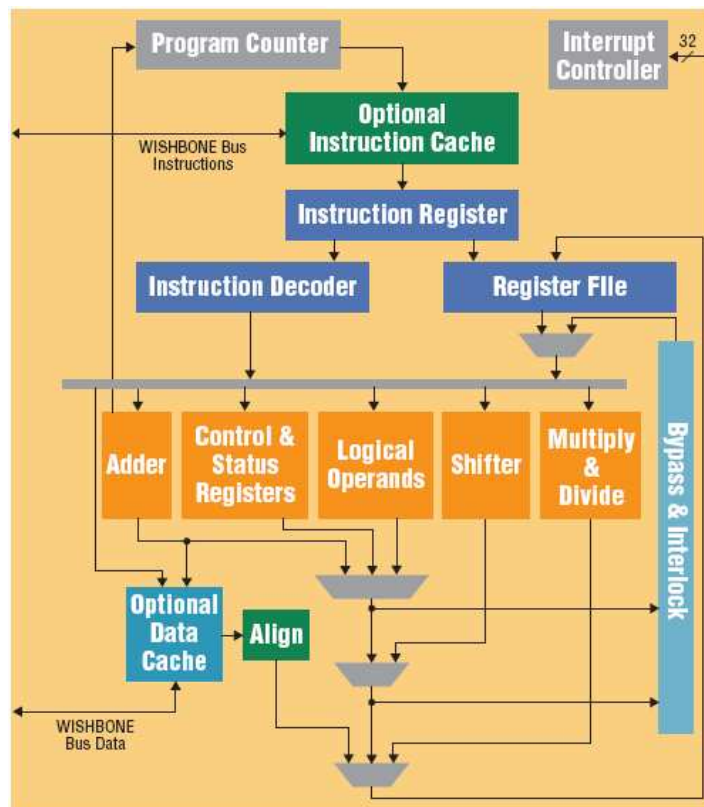
The aeMB core is an attempt of the open source community to copy the architecture of the Xilinx Micro Blaze core. It was possible to synthesize this design but it did not work on the Xilinx evaluation board. The design was synthesized with and without optimisation. The designer of this core did actually implement it on a Xilinx evaluation board. Despite this, it was not possible to get the core working, and due to time constraints it was decided to move on.

The Openrisk, Leon 3 and Aquarius were too big to be considered for the design. Very little information was offered on the Aquarius, therefore it was necessary to synthesize the core to determine the size.



The only other small core left is the LM32 core, and it was evaluated last. This core is a very well documented open source core actively maintained by Lattice. It comes with a plug-in for the Eclipse IDE. Further more a whole suite of peripherals are offered with it. Most of the peripherals are from the opencores.org web site, but they have been tested on the LM32 architecture. What is also worth noting is that Lattice offers a scaled down 8-bit version of the LM32. uClinux and uCOS ports are also offered with the core. Each RTOS was very well documented.

### Section 4.1.2.1 LM32 soft-core architecture



**Figure 25. LM32 block diagram**

Figure 25 shows the block diagram of the LM32 soft-core. The processor uses a 32-bit 6-stage pipeline with bypass and interlock logic. The six pipeline stages are:

- Address – The address of the instruction to execute is calculated and sent to the instruction cache.
- Fetch – The instruction is read from memory.

- Decode – The instruction is decoded, and operands are either fetched from the register file or bypassed from the pipeline.
- Execute – The operation specified by the instruction is performed. For simple instructions such as addition or a logical operation, execution finishes in this stage, and the result is made available for bypassing.
- Memory – For more complicated instructions such as loads, stores, multiplies, or shifts, a second execution stage is required.
- Write back – Results produced by the instructions are written back to the register file.

The processor has 32 32-bit registers. The first eight functional parameters are passed in the registers and the remaining arguments are placed on the stack.

The LM32 has an optional data and instruction cache, but for this design the cache is not enabled since the performance increase is not required. The LM32 has a flat byte addressable 32-bit address space that uses the big-endian standard (MSB is at lowest address).

All memory accesses must be aligned to the size of the access. No check is performed for unaligned access and it will result in undefined behaviour. 8-bit, 16-bit and 32-bit aligned access is allowed. The stack grows towards higher memory.

The LM32 architecture supports both software and hardware breakpoints. Software breakpoints should be used for setting breakpoints in code, which resides in volatile memory such as DDR or SRAM, while hardware breakpoints should be used for setting breakpoints in code, which resides in non-volatile memory, such as FLASH or ROM.

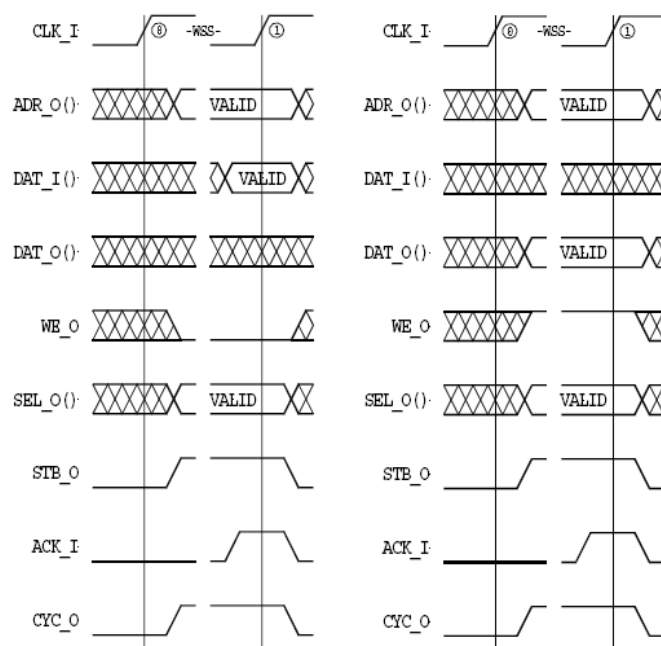
### **Section 4.1.3 Peripherals**

The LM32 core contains a host of peripherals mostly from the [www.opencores.org](http://www.opencores.org) web site. In most instances these peripherals were designed with extensive feature set which makes them big. It was decided to create new small peripherals with limited features written in VHDL.

The LM32 soft core is written in Verilog. The author is fluent in VHDL therefore the high level instantiation of the core is done in VHDL, and all peripherals are written in VHDL with test benches written for Modelsim.

The following peripherals were developed for this project: external memory bus, general purpose I/O, 32-bit timer with PWM, SPI bus and UART.

The LM32 core is build on the Wishbone bus architecture (see [www.opencores.org](http://www.opencores.org)). Wishbone is an open source bus standard, and functions as a common interface to all IP cores and memories. A Wishbone master always talks to a Wishbone slave. The master is responsible for generating the bus cycle while the slave receives the bus cycle.



**Figure 26. Wishbone read and write bus cycle**

Figure 26 shows the Wishbone bus cycle. A data read takes place on the rising edge of the clock while STB\_O and ACK\_I is high and WE\_O is low. The data write takes place on the rising edge of the clock while STB\_O, ACK\_I and WE\_O is high. ACK\_I is an asynchronous acknowledge signal.

SEL_O[3:0]	Byte selection
1000	MSB
0100	
0010	
0001	LSB

**Table 3. Wishbone SEL\_O byte map**

The LM32 is a big Endian core and Table 3 shows how SEL\_O maps to the corresponding byte within the 32-bits. Bus address lines A1 and A0 are not used by the core, and SEL\_O is used for BYTE and WORD selection. Table 4 shows the byte location of the data bus.

WBS_DAT_I/WBS_DAT_O	Byte definition
[31:24]	MSB
[23:16]	
[15:8]	
[7:0]	LSB

**Table 4. Wishbone data in and out bus definition**

### Section 4.1.4 HDL design

The LM32 has a separate bus for data and code and each of these is a Wishbone master. All peripherals and memory connected to these busses are Wishbone slave devices. For flexibility the hardware design has a data bus, instruction bus and a peripheral bus. It was decided to separate the peripheral bus from the data bus to eliminate bus stalling. Experiments showed long bus delays on the data bus when combined with slow peripherals.

To simplify the interfacing of the peripherals to the core and the debug architecture, a bus arbiter was designed to combine the instruction, peripheral and data bus into one shared bus. A shared bus arbiter is used to combine the instruction and data bus into one bus. Both the instruction and data bus are bus masters, and the arbiter will connect the single shared bus to a specific bus master if it is not used by the other bus master. All peripherals and memories connect through this single shared bus to the processor core.

In future, this simplification can be removed if better performance is required.

The design makes provision for a debug and operational processor mode. In debug mode the debug ROM kernel is mapped to the reset vector to ensure execution after POR. It is only through the execution of this kernel that the LM32 can talk to the GDB debugger on the PC.

The addressable area is broken up in 8KB pages and address lines A0 – A12 are ignored in address decoding. Even though this is a Harvard architecture none of the areas between the instruction and data bus overlaps. This was a deliberate decision to simplify the linker file as GCC does not inherently support Harvard architecture. All flash and ROM areas are

accessible through the bus arbiter. This is done to support in system programming and debugging.

Address range 0 to 0x7FFFFFFF are reserved for cached area leaving address area 0x80000000 to 0xFFFFFFFF for peripherals.

Debug:			Normal:		
0xFFFFD000	I2C	8KB	0xFFFFD000	I2C	8KB
0xFFFFB000	IO 1	8KB	0xFFFFB000	IO 1	8KB
0xFFFF9000	IO 2	8KB	0xFFFF9000	IO 2	8KB
0xFFFF7000	UART 1	8KB	0xFFFF7000	UART 1	8KB
0xFFFF5000	UART 2	8KB	0xFFFF5000	UART 2	8KB
0xFFFF3000	TIMERS	8KB	0xFFFF3000	TIMERS	8KB
0xFFFF1000	SOC registers	8KB	0xFFFF1000	SOC registers	8KB
0xFFFEF000	SPI	8KB	0xFFFEF000	SPI	8KB
0xFFFED000	H-Bridge 1	8KB	0xFFFED000	H-Bridge 1	8KB
0xFFFEB000	H-Bridge 2	8KB	0xFFFEB000	H-Bridge 2	8KB
0x031FFFFF	Debug mode ROM: SRAM @ instruction bus	1MB	0x031FFFFF	SRAM @ data bus	2MB
0x03100000			0x03000000		
0x030FFFFF	Debug mode RAM: SRAM @ data bus	1MB	0x01FFFFFF	Flash @ instruction bus	16MB
0x03000000			0x01000000		
			0x0000E000		
			0x0000C000		
			0x0000A000		
			0x00009FFF		
0x01FFFFFF	Flash @ instruction bus	16MB	0x00008000	On Chip RAM @ data bus	8KB
0x01000000			0x00007FFF	Boot loader/ROM monitor	8KB
0x0000E000			0x00006000	On Chip ROM	8KB
0x0000C000			0x00005FFF		
0x0000A000			0x00004000		
0x00009FFF	On Chip RAM @ data bus	8KB	0x00003FFF		
0x00008000	On Chip ROM	8KB	0x00002000		
0x00007FFF			0x00001FFF		
0x00006000			0x00000000		
0x00005FFF					
0x00004000					
0x00003FFF					
0x00002000					
0x00001FFF	ROM monitor	8KB			
0x00000000					

**Figure 27. CPU address map for different modes**

Figure 27 shows the processor address map for normal and debug mode. The ROM monitor in debug mode contains the GDB stub implementation (see Section 4.1.7.1 ).

### Section 4.1.4.1 Peripherals

At the moment all the peripherals run at the same clock frequency as the processor.

#### *I/O Port*

To accept input and control actuators the core requires general purpose input and output ports. The port design is bidirectional and bit addressable and multiple instances can be instantiated. Table 5 shows the component definition for the I/O port. The SOC contains 2 I/O ports.

component IO	
Port (ADR_I	: in std_logic_vector(3 downto 0);
DAT_I	: in std_logic_vector(7 downto 0);
DAT_O	: out std_logic_vector(7 downto 0);
CLK_I	: in std_logic;
RST_I	: in std_logic;
ACK_O	: out std_logic;
STB_I	: in std_logic;
WE_I	: in std_logic;
CYC_I	: in std_logic;
--Non wishbone signals	
HW_PORT_PINS	: inout std_logic_vector(7 downto 0));
end component;	

**Table 5. I/O Port VHDL component definition**

#### *Timer*

Timers are also important especially when using a RTOS in which case the operating system tick is generated by a time overflowing periodically. An array of timers was created in VHDL and during instantiation the required timer count (N\_TIMERS) is specified. The width of the timer (TIMER\_WIDTH) is also specified and the default value is 32-bits. Optionally the timer can be configured as a 32-bit Pulse Width Modulator (PWM). Table 6 shows the component definition for the Timer. The SOC contains 4 timers of which one is dedicated to the RTOS.

```

component Timers
generic( N_TIMERS      : integer range 1 to 8 := 1;
        TIMER_WIDTH    : integer range 8 to 32 := 32 );
Port ( CLK_I      : in  STD_LOGIC;
       RST_I      : in  STD_LOGIC;
       ADR_I      : in  STD_LOGIC_VECTOR (31 downto 2);
       DAT_I      : in  STD_LOGIC_VECTOR ((TIMER_WIDTH-1) downto 0);
       DAT_O      : out STD_LOGIC_VECTOR ((TIMER_WIDTH-1) downto 0);
       ACK_O      : out STD_LOGIC;
       STB_I      : in  STD_LOGIC;
       SEL_I      : in  STD_LOGIC_VECTOR (3 downto 0);
       CYC_I      : in  STD_LOGIC;
       WE_I      : in  STD_LOGIC;

       timers_out   : out std_logic_vector((N_TIMERS-1) downto 0);
       timers_pwm_out : out std_logic_vector((N_TIMERS-1) downto 0));
end component;

```

**Table 6. Timer VHDL Component definition**

## ***UART***

To communicate to peripherals like a wireless modem and GPS receiver the core requires a serial port. A lightweight UART was designed specifically for this task and to use as a debug port. The UART design can be instantiated multiple times. The current implementation contains 4 UART's. Table 7 shows the component definition for the UART.

The baud generator of the UART does not need a specific clock frequency like 1.8432MHz but can work of the CPU frequency of 32MHz. Normally the divisor for 115200 baud at 32MHz would be 277.78. To achieve a stable baud rate we sometimes divide by 277 and sometimes by 278, through the cycle, this average out to 277.78. Table 8 shows the VHDL implementation for the baud generator.

```

component serial
generic ( BAUD_RATE   : integer;
          CORE_FREQ_KHZ : integer);
Port ( WBS_ADR_I   : in std_logic_vector(3 downto 2);
        WBS_DAT_I   : in std_logic_vector(31 downto 0);
        WBS_DAT_O   : out std_logic_vector(31 downto 0);
        WBS_CLK_I   : in std_logic;
        WBS_RST_I   : in std_logic;
        WBS_ACK_O   : out std_logic;
        WBS_STB_I   : in std_logic;
        WBS_WE_I    : in std_logic;
        WBS_SEL_I   : in std_logic_vector(3 downto 0);
        WBS_CYC_I   : in std_logic;

        serial_intr  : out std_logic;

        -- Serial port Interface
        txpin        : out std_logic;
        rxpin        : in std_logic);
end component;

```

**Table 7 UART VHDL Component definition**

```

baud_inc = ((Baud<<(BaudGeneratorAccWidth-4))+(ClkFrequency>>5))/(ClkFrequency>>4);

process (WBS_CLK_I, WBS_RST_I)
begin
    if WBS_RST_I = '1' then
        baud_inc <= conv_std_logic_vector(gen_inc_val,(ACC_WIDTH+1));
        baud_accum <= (others => '0');
    elsif rising_edge(WBS_CLK_I) then
        if baud_inc = conv_std_logic_vector(0,ACC_WIDTH) then
            baud_inc <= conv_std_logic_vector(gen_inc_val,ACC_WIDTH+1);
        elsif divreload = '1' then
            baud_inc <= data_in;--WBS_DAT_I;
            baud_accum <= (others => '0');
        end if;
        baud_accum <= ('0' & baud_accum((ACC_WIDTH-1) downto 0)) + baud_inc;
    end if;
end process ;
baudx16_clk <= baud_accum(ACC_WIDTH);

```

**Table 8. Baud generator code**



## Memory

The BRAM inside the FPGA is grouped together in 2KB x 32-bit blocks and form the building elements for ROM and RAM storage. The only difference between ROM and RAM is write access. Table 9 shows the definition of each component.

<pre> component BRAM_2K_32bits_WB port(     CLK_I    : in std_logic;     RST_I    : in std_logic;     ACK_O    : out std_logic;     STB_I    : in std_logic;     CYC_I    : in std_logic;     WE_I     : in std_logic;     DAT_I    : in std_logic_vector(31 downto 0);     DAT_O    : out std_logic_vector(31 downto 0);     SEL_I    : in std_logic_vector(3  downto 0);     ADR_I    : in std_logic_vector(12 downto 2)); end component; </pre>	<pre> component BROM_2K_32bits_WB port(     CLK_I    : in std_logic;     RST_I    : in std_logic;     ACK_O    : out std_logic;     CYC_I    : in std_logic;     STB_I    : in std_logic;     DAT_O    : out std_logic_vector(31 downto 0);     ADR_I    : in std_logic_vector(12 downto 2)); end component; </pre>
--	---

**Table 9. Onboard ROM/RAM VHDL Component definition**

When the FPGA goes through it's configuration cycle the BRAM blocks are initialised with data from the bitmap file. It is possible to add the application binary data in the FPGA bit file (see data2mem in Section 4.1.7.1 ) so that the BRAM (simulated ROM) contains the application after FPGA configuration. Once the FPGA configuration is completed the soft processor core can start executing the application binaries.

The design also caters for external flash and ram through a 16-bit external instruction and data bus. This bus is connected to the 32-bit wishbone bus through a separate RAM/Flash controller. This controller can include wait states (COUNT\_CYCLE\_EXTEND) as required. It supports byte, word and double word read and write operations.

For each external access the controller does two consecutive external bus cycles, thereby increasing the external bus access time by a factor of two. The power consumption requirement has a higher priority than performance. A 32-bit external bus would require 2 additional components that would increase the total power consumption. Table 10 shows the VHDL component definition for the external memory bus interface.

COMPONENT SRAM		
generic(	COUNT_CYCLE_EXTEND	: integer := 1;
	WORD_ADR_WIDTH	: integer := 20 );
port (	CLK_I	: std_logic;
	RST_I	: std_logic;
-- Wishbone slave interface		
	ACK_O	: inout std_logic;
	STB_I	: in std_logic;
	CYC_I	: in std_logic;
	WE_I	: in std_logic;
	DAT_I	: in std_logic_vector(31 downto 0);
	DAT_O	: out std_logic_vector(31 downto 0);
	SEL_I	: in std_logic_vector(3 downto 0);
	ADR_I	: in std_logic_vector(WORD_ADR_WIDTH downto 2);
--RAM chip interface		
	ADR	: out std_logic_vector(WORD_ADR_WIDTH downto 1);
	DQ	: inout std_logic_vector(15 downto 0);
	nWE	: out std_logic;
	nOE	: out std_logic;
	nUB	: out std_logic;
	nLB	: out std_logic;
	nCE	: out std_logic );
END COMPONENT;		

**Table 10. External memory bus VHDL component definition.**

## ***SPI Bus***

The SPI bus is a low pin-count interconnection bus and was chosen as the main interface to peripheral devices. The HDL design caters for all variations like clock phase, clock polarity, delay time and shift direction. Each SPI port can only function as a master device. The SOC contains one SPI port but makes provision for a second SPI port. Table 11 shows the SPI component definition. SHIFT\_DIRECTION specifies whether the most or least significant bit is send first. CLOCK\_PHASE specifies if data is latched on the leading or trailing edge of the clock. CLOCK\_POLARITY specifies the idle logic stat of the clock line.

COMPONENT spi	
generic(	SHIFT_DIRECTION : integer := 0;
	CLOCK_PHASE : integer := 0;
	CLOCK_POLARITY : integer := 0;
	SLAVE_NUMBER : integer := 1;
	DATA_LENGTH : integer := 3;
	DELAY_TIME : integer := 2;
	INTERVAL_LENGTH : integer := 2);
PORT(	CLK_I : IN std_logic;
	RST_I : IN std_logic;
	SPI_ADR_I : IN std_logic_vector(31 downto 0);
	SPI_DAT_I : IN std_logic_vector(31 downto 0);
	SPI_DAT_O : OUT std_logic_vector(31 downto 0);
	SPI_ACK_O : OUT std_logic;
	SPI_STB_I : IN std_logic;
	SPI_SEL_I : IN std_logic_vector(3 downto 0);
	SPI_CYC_I : IN std_logic;
	SPI_WE_I : IN std_logic;
	SPI_CTI_I : in std_logic_vector(2 downto 0);
	SPI_BTE_I : in std_logic_vector(1 downto 0);
	SPI_LOCK_I : in std_logic;
	SPI_ERR_O : out std_logic;
	SPI_RTY_O : out std_logic;
	MISO_MASTER : IN std_logic;
	MOSI_MASTER : OUT std_logic;
	SS_N_MASTER : OUT std_logic_vector((SLAVE_NUMBER-1) downto 0);
	SCLK_MASTER : OUT std_logic;
	SPI_INT_O : OUT std_logic;
END COMPONENT;	

**Table 11. SPI VHDL Component definition**

### ***H-Bridge controller***

The H-bridge controller is designed to drive a DC motor with the following modes: forward speed control, reverse speed control, braking and stop. The speed and level of breaking is configurable through the duty cycle register. The period of the PWM signal is also configurable. The SOC contains 2 H-Bridge controllers.

```

entity H_bridge is
  Port ( CLK_I   : in  STD_LOGIC;
        RST_I   : in  STD_LOGIC;
        ADR_I   : in  STD_LOGIC_VECTOR (31 downto 2);
        DAT_I   : in  STD_LOGIC_VECTOR (31 downto 0);
        DAT_O   : out STD_LOGIC_VECTOR (31 downto 0);
        ACK_O   : out STD_LOGIC;
        STB_I   : in  STD_LOGIC;
        SEL_I   : in  STD_LOGIC_VECTOR (3 downto 0);
        WE_I    : in  STD_LOGIC;

        -- Due to the hardware design, highside driver is inverted.
        n_left_top_plus      : out std_logic;
        left_bottom_plus     : out std_logic;
        n_right_top          : out std_logic;
        right_bottom         : out std_logic);
end H_bridge;

```

**Table 12. H-Bridge VHDL Component definition**

## *Quadrature decoder*

A quadrature device is used to sense position and rotation by converting the displacement into digital pulses. The phase difference between output signal A and output signal B determines the direction of rotation. For example, if pulse output A leads pulse output B the shaft is rotating in the clockwise direction.

```

component quad_decoder is
  generic( QUAD_COUNT_WIDTH : integer := 32 );
  port( --Wishbone bus signals
        wb_clk_i   : in  std_logic;
        wb_rst_i   : in  std_logic;
        wb_stb_i   : in  std_logic;
        wb_cyc_i   : in  std_logic;
        wb_ack_o   : out std_logic;
        wb_adr_i   : in  std_logic_vector(1 downto 0); --assumes 32 bit alignment
        wb_dat_o   : out std_logic_vector(31 downto 0);
        wb_dat_i   : in  std_logic_vector(31 downto 0);
        wb_we_i   : in  std_logic;

        --Quadrature inputs
        quad_cha_i : in  std_logic; --Quadrature channel A
        quad_chb_i : in  std_logic; --Quadrature channel B
        quad_idx_i : in  std_logic; --Quadrature index
        quad_lat_i : in  std_logic; --Quadrature latch cnt input
        quad_irq_o : out std_logic --Quadrature IRQ out

        );
end component;

```

**Table 13. Quadrature decoder VHDL component definition**

A quadrature decoder takes the output signals (A, B, and Index) from the quadrature encoder as inputs and converts these signals into a numerical value that can be used to determine position, distance, velocity, and other functions. Table 13 shows the definition for the VHDL component. The SOC contains two quadrature decoders.

#### **Section 4.1.4.2 Clocks**

Inside the FPGA the delayed-locked loop of the DCM is used to remove any clock skew. The clock is then split into two paths each going into a programmable divider. The one path goes to the LM32 soft core while the other one is used for the peripheral clock. There is also an enable/disable bit for the peripheral clock. Each output goes through a BUFG buffer specifically design to buffer and gate clock resources inside the FPGA.

#### **Section 4.1.4.3 Power management**

A FPGA based design is usually not a low power platform. Power management aims to limit the power consumption by implementing intelligent power control algorithms. When a peripheral is not used it is placed in a sleep mode and its clock is disabled. This reduces the power consumption since there are no signal transitions in the peripheral due to the absence of the clock.

The clock to a peripheral is disabled by using a BUGCE or a BUFGMUX primitive. These are resources inside the FPGA that were designed by the manufacturer to select and gate clocks. If not explicitly specified, the FPGA will use a multiplexer for the enable to the CLB that will not disable the clock to the flip flops inside the CLB. Thus the clock will continue to charge and discharge the input capacitance of the flip flop.

The power management also controls the LM32 core clock frequency, and when high processing speed is not required the clock frequency is scaled down. The LM32 core clock frequency is also scaled at power-up. After POR the LM32 runs at a reduced clock frequency placing a smaller demand on the power supply. This was only done for a brief moment after which the clock frequency is scaled back to the full value.

The FPGA core temperature is monitored in an attempt to reduce the static power consumption. This is done through a diode inside the FPGA silicon. The temperature is used

to scale the LM32 core clock frequent. When the temperature is too high the clock frequency is reduced, but when it is under a threshold value the clock frequency is not scaled.

#### **Section 4.1.4.4 Dynamic Partial Self Reconfiguration**

The design is partitioned into blocks. There is one fixed block containing the LM32 soft-core and the ICAP interface. This block will never change dynamically but will always initiate the dynamic reconfiguration of the other blocks (see Figure 29). The layout of the blocks is done in PlanAhead which export a user constraint file (UCF). The UCF file is imported into the ISE project containing the soft-core and all the VHDL code for each block. The ISE project must also contain the clock network which cannot change during a partial reconfiguration.

#### **Section 4.1.4.5 Floorplanning**

Floorplanning the design proved to be a very difficult step in the design process. There are conflict priorities and the optimal floor plan is a compromise between partial reconfiguration layout, IOB's voltage compatibility and peripheral location on the PCB.

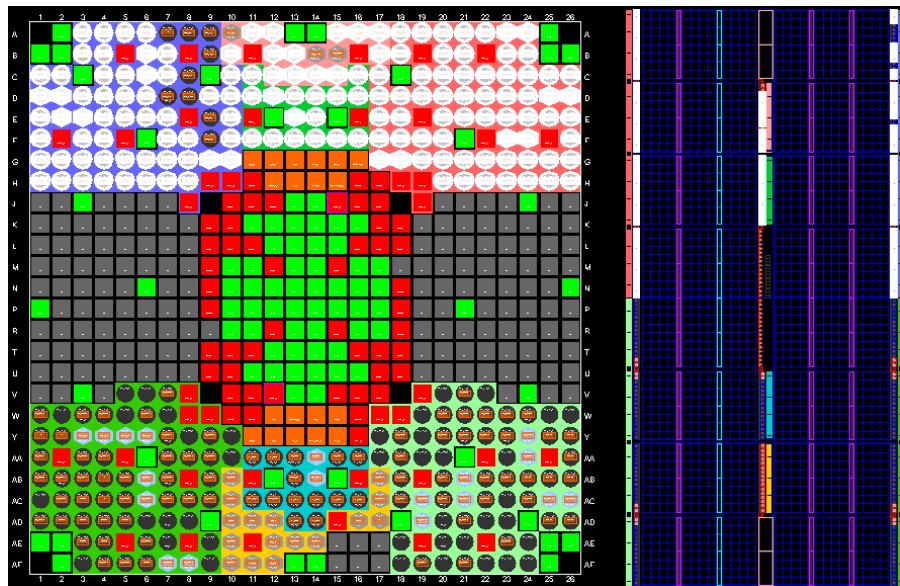
To improve signal integrity it is important to have the peripheral pins as close to the actual peripherals as possible. This reduces the track length and track impedance. The problem arises when there are multiple peripherals and different external busses. This design has a data bus, instruction bus, peripheral bus and numerous peripherals. All the peripherals were grouped in an essential core group and a reconfigurable group. The core peripherals and the CPU busses should all fit inside the fixed area (partial reconfigurable design should have a fixed area). This area should be as small as possible to ensure that a large area is reserved for the partial reconfigurable blocks, and if possible, should not cover multiple clock regions as these define the minimum size of the PR blocks.

To reduce the power consumption all IOB's are used at the lowest possible voltage. This voltage is dictated by the peripherals connected to it. There are two voltages used: 3.3V and 2.5V. Similar voltage peripherals and busses are grouped together and placed on IOB's. There are some instances where it was not possible to do this, then level shifting was done to ensure proper operation.

The ICAP port has two fixed locations inside the Virtex die, one at the top and one at the bottom. The PR fixed block should contain one of these sites. It was also decided to allow software applications running on the soft core to access the configuration PROM after FPGA configuration. The SelectMAP port was used to access the configuration PROM and should be as close as possible to the fixed PR block.

As a first iteration the FPGA die was divided into two sections; one on the left and one on the right. It was very difficult to fit all the peripherals and busses into the fixed block and some signals had to go through level shifting. After completing the floor plan a WASSO analyses was done and numerous banks either failed or was on the limit. This approach was abandoned.

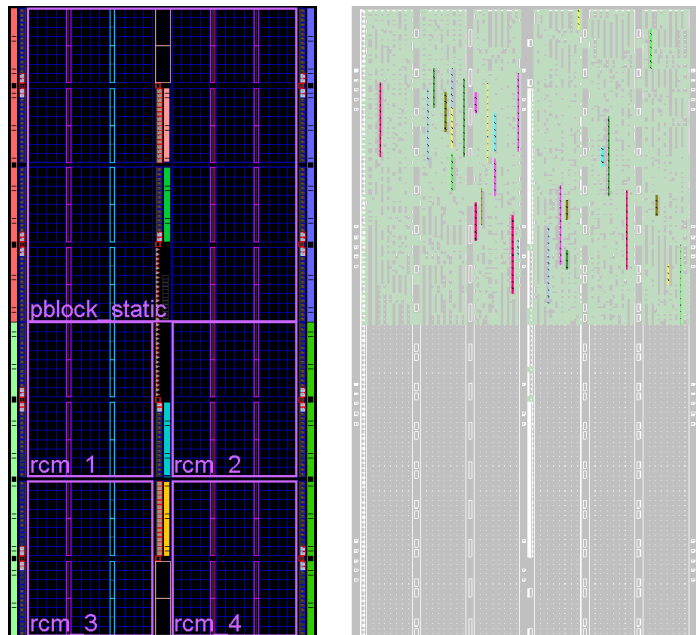
The second iteration also divided the FPGA die into two areas but they were at the top and at the bottom. All the factors discussed above were taken into consideration and the peripherals and busses were assigned. Fewer level shifting was required and the WASSO analysis passed. Figure 28 shows the FPGA top view and the die. The white highlighted pins are the core signals present in the fixed module that contains the soft-core.



**Figure 28. Processor & peripherals floorplan**

Another advantage of this layout is the ability to define four almost homogeneous PR blocks and the fixed PR block contains considerable more RAM than the previous approach.

Figure 29 shows the static block (pblock\_static) containing the LM32 core with its peripherals and 4 partial reconfigurable blocks (rcm 1 to 4).



**Figure 29. LM32 PBLOCK areas and floor planning**

The right most image in Figure 29 shows the LM32 core with peripherals fitted inside the pblock\_static area.

### Section 4.1.5 Power consumption optimisation

This section looks at static power analysis and does not take into count the dynamics of the software running on the soft core. Table 14 shows the estimated power consumption for the soft processor core and all its peripherals. The analysis was done using Xilinx's XPower tool and it clearly shows that most of the power consumed is done by the internal logic and leakage. The IOB's consumption is very low (see Vcco33).

Name	Value
Clocks	0.07600 (W)
Logic	0.00812 (W)
Signals	0.02797 (W)
IOs	0.14025 (W)
BRAMs	0.00859 (W)
DCMs	0.00000 (W)
DSPs	0.00065 (W)
Total Quiescent Power	0.25834 (W)
Total Dynamic Power	0.26157 (W)
Total Power	0.51990 (W)
Junction Temp	35.9 (degrees C)

Name	Power (W)	Voltage	Range	Icc (A)	Iccq (A)
Vccint	0.18653	1.200	1.140 to 1.260	0.10110	0.05434
Vccaux	0.20145	2.5		0.00333	0.07725
Vcco33	0.01884	3.3		0.00571	0.00000
Vcco25	0.11308	2.5		0.04523	0.00000

**Table 14. FPGA power analysis without power optimisation**



After numerous attempts in changing the design implementation options in the Xilinx ISE tool the optimised results is shown in Table 15. The only real difference is in the 1.2V supply (Vccint). This can be contributed to the changing in communication primitives as described in Section 3.4.3 .

Name	Value
Clocks	0.07760 (W)
Logic	0.00350 (W)
Signals	0.01911 (W)
IOs	0.14091 (W)
BRAMs	0.00843 (W)
DCMs	0.00000 (W)
DSPs	0.00065 (W)
Total Quiescent Power	0.25809 (W)
Total Dynamic Power	0.25021 (W)
Total Power	0.50830 (W)
Junction Temp	35.7 (degrees C)

Name	Power (W)	Voltage	Range	Icc (A)	Iccq (A)
Vccint	0.17427	1.200	1.140 to 1.260	0.09108	0.05414
Vccaux	0.20149	2.5		0.00335	0.07725
Vcco33	0.01947	3.3		0.00590	0.00000
Vcco25	0.11308	2.5		0.04523	0.00000

**Table 15. FPGA power analysis with power optimisation**

At this stage it is unclear why the improvement is only marginal. Initially it was thought that the low clock frequency might be responsible for the marginal improvement but after changing it from 32MHz to 80MHz the same results was observed. At least for this design the Xilinx power optimisation has a small effect on power consumption.

Name	Power (W)*
Hierarchy total	0.05030
soc	0.00897 / 0.05030
ext_flash	0.01525
uart2	0.00280 / 0.00562
rom0	0.00135
ram	0.00221
rom2	0.00250
rom_monitor	0.00251
rom1	0.00253
uart1	0.00281 / 0.00562
io_port	0.00033
io_port2	0.00034
spi1	0.00076
ext_sram	0.00081
cpu_core	0.00005 / 0.00104
H_bridge1	0.00018
H_bridge2	0.00018
shared_bus_arbiter	0.00004
timers	0.00000 / 0.00005

**Table 16. SOC Power analysis by hierarchy**

Table 16 shows the hierarchy of the power consumption. The external data and instruction bus contributes 16.06 mW. In the next section a configuration mode is discussed that does not require external flash or RAM thereby reducing the static consumption by 16.06 mW. In practice this saving would increase due to frequent access to the external bus while executing the application software.

### Section 4.1.6 Hardware Design

The main board, shown in Figure 30, forms the heart of each agent with different sub-modules interfacing to this board. This board is responsible for all data processing and decision making, and utilises the Xilinx Virtex 4 device that supports dynamic self partial reconfiguration. The hardware design supports the Virtex 4 LX15, LX25 and FX12 devices.

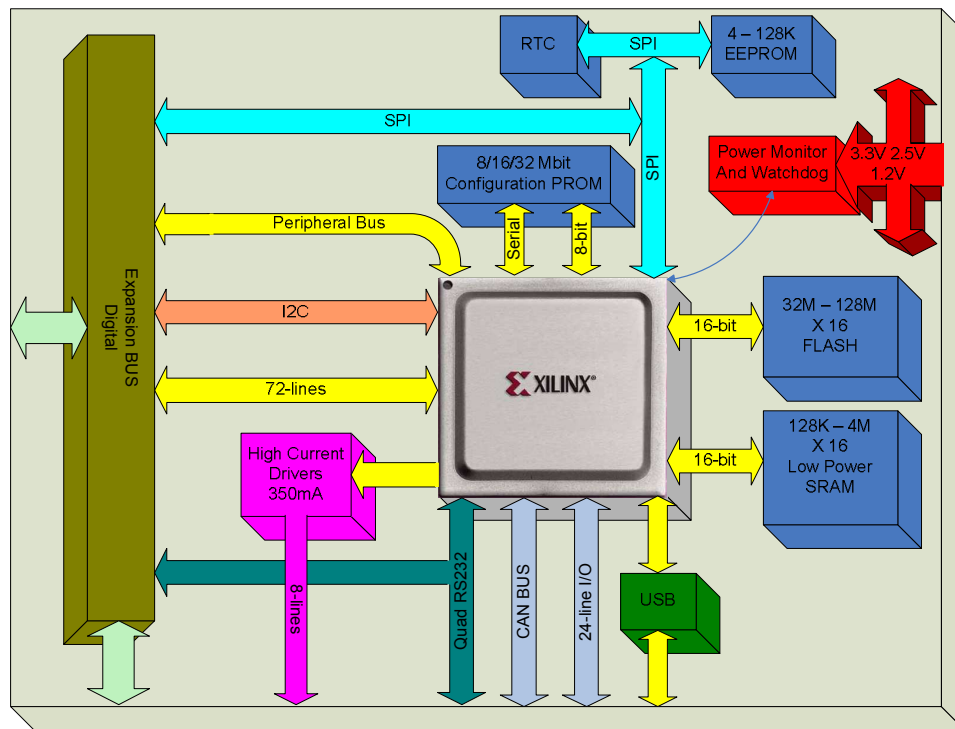


Figure 30. Main board block diagram

The Xilinx parallel configuration PROM was chosen above the serial PROM to improve the dynamic response of partial reconfiguration. The Virtex 4 device supports clock speeds of up to 100MHz to the 8-bit parallel configuration PROM. This means that a complete FPGA reconfiguration on the Virtex 4 LX25 will take less than 10 ms. The parallel PROM also supports compression, which almost double the storage capacity of the PROM. Master and slave serial and parallel configuration modes are supported.

The hardware supports two configurations: 1) A minimal configuration which does not require external RAM or flash. 2) A full configuration utilising the external RAM and flash.

The minimal configuration implementation is used to reduce the implementation complexity, and to further reduce power consumption by eliminating additional external components (see

Table 16 in Section 4.1.5 ). The boot loader binary image is merged into the FPGA configuration bit stream by using a Xilinx software tool. The application software binary image is stored inside the configuration PROM above the FPGA configuration bit stream. After completion of the FPGA configuration the soft-core will start executing the boot loader, which in turn will read the application image from the configuration PROM and write it to block RAM inside the FPGA. When this is done the boot loader will jump to the start address and start executing the application code. Obviously there is a limit to the size of the application code, since the Virtex 4LX25 contains only 1296 Kbits of block RAM of which one block is reserved for the boot loader code. The Virtex 4LX15 contains 864 Kbits block RAM.

The second configuration is supported to accommodate large designs that utilises an operating system like embedded Linux. A generic footprint is used for the flash and the RAM, thereby supporting different size RAM and Flash devices from different manufacturers. The following devices are chosen because of their densities and low power consumption:

Micron MT45W4MW cellular RAM

ST M58LW064A flash

Both these devices expose a SRAM type interface that ensures a simple memory interface to the soft-core.

### **Section 4.1.6.1 Component selection**

The main board was designed with low power usage in mind.

Each of the four UARTS goes through a RS232 level translator IC. The chosen RS232 transceiver has the ability to automatically shutdown to reduce power consumption (a typical RS232 transceiver can draw as much as 25mA). It can detect a valid voltage level on the RS232 side, and automatically wake up or notify the microprocessor which in turn will enable the transceiver.

The CAN bus transceiver used is from Texas Instruments and works from 3.3V opposed to the other devices that requires 5V.

The USB transceiver has a suspend mode to limit the power consumption when not in use.

For the external RAM it was decided to use static RAM instead of dynamic RAM. The dynamic RAM must be refreshed periodically. Each refresh cycle consumes power while the SRAM chip is in sleep mode when not accessed and only consumes energy when accessed. To further reduce the consumption an ultra low power SRAM device was chosen from BSI. This device can operate on 2.4 – 5.5V but on the main board it is connected to 2.5V and to a 2.5V bank on the FPGA. Each read and write action must charge the gate capacitance of the RAM and FPGA. The energy required to do this is quadratic related to the operational voltage. Thus, reducing the supply voltage has a profound effect on the dynamic power consumption.

Finding a Flash that can operate at 2.5V with a popular PCB footprint prove to be difficult. The only device that falls within this requirement is the P33 StrataFlash from Numonyx. The P33 is pin-compatible with numerous common flash devices. Provision was also made to power different voltage flash devices up to 3.3V.

#### **Section 4.1.6.2 Core temperature sensing**

The Virtex 4 has a temperature diode embedded into the FPGA that is used to measure the core temperature. This diode is connected to a temperature sensor that digitises the temperature readings. This reading is monitored by a state machine inside the FPGA that scales the clock frequency accordingly. By doing this the core temperature of the FPGA is held within predefined limits thereby reducing the static power consumption (leakage current).

The Texas Instruments TMP401 was chosen due to its 12-bit resolution and  $\pm 1^{\circ}\text{C}$  remote sensing accuracy. The two wire serial (I2C) interface simplifies the temperature controller design inside the FPGA. If required a trip point can be set through the I2C interface and when reached the TMP401 will interrupt the processor through one of its external interrupt lines.

#### **Section 4.1.6.3 Interfaces**

##### ***Internal***

The main board contains an expansion bus (see Figure 30) that consists of an 80-pin and a 40-pin connector. The 80-pin connector exposes a general-purpose bus that is reserved for

future use. This includes I/O lines to each of the reconfigurable modules shown in Figure 29. It also contains two RS232 ports. All the lines on this connector have a serial resistor for short circuit protection, but they have no over voltage protection. This connector is intended for large scale expansion such as the peripheral board in Section 4.3 .

The 40-pin connector contains a buffered peripheral bus used to expand the peripherals of the soft-core (A2D, D2A, motor controller). The peripheral bus is a Wishbone master bus. It also contains an I2C and SPI bus.

On the main board the SPI bus is used to connect the real-time clock and EEPROM to the micro controller.

### ***External***

Two RS232 serial ports are reserved for external communication. When a wireless modem is used at a later stage it will connect to one of these serial ports.

On the main board provision is made for a master or slave USB port. Once this port is implemented inside the FPGA it will be used for debugging and software downloading.

#### **Section 4.1.6.4 FPGA configuration**

The main board supports different modes for configuring the FPGA. The most popular ones are serial configuration and parallel configuration through the SelectMAP port. The parallel interface is 8-bit wide and is the preferred configuration mode. A slave mode is also supported where the configuration PROM is driving the configuration clock. When using this mode it is possible to compress the bit stream, which almost doubles the storage capacity of the configuration PROM.

Because of the fast rise and fall times of the clock it must have proper termination. R29 and R30 provide this termination and 100 ohm resistors are used (see Figure 31). There is no minimum start-up time requirement and therefore it is not necessary to run at 100MHz. Working at lower clock speeds has two power reduction advantages: the low value termination resistors can be increased or possibly removed, and the lower clock speed means slower charging and discharging of gate capacitance.

It is useful to store data or program binaries in the configuration PROM. This can be used instead of the external Flash to further reduce power consumption. This design utilise the FS48/FSG48 package that has an upgrade path to a 32 Mbit device. This leaves enough room for user data/applications. After the FPGA is configured the configuration port is disabled and the clock is stopped effectively disabling the configuration PROM. To utilise this storage after the initial configuration, the hardware makes provision for the soft processor core to drive the clock, data and chip enable of the PROM. This is accomplished by routing user I/O pins to the corresponding configuration pins. While the initial configuration is in progress, these user I/O pins are in 3-state.

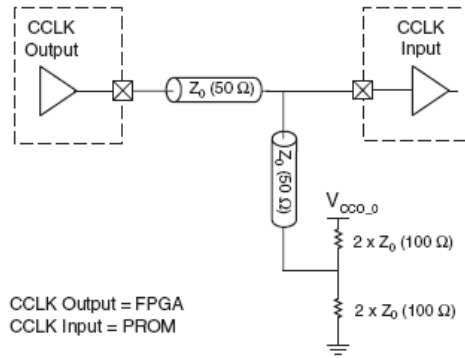
The following pins on the PROM are under the soft-core control after initialisation: clock (CLK), enable (#CE), output enable/reset (OE/#RESET) and the 8-bit data bus (D0-D7) or D0 if used in serial mode. The data bus (D0 – D7) is dual purpose I/O but the other signals are dedicated.

#### **Section 4.1.6.5 PCB**

The Virtex 4 device can clock the configuration PROM at speeds of up to 100MHz. The configuration I/Os use the LVCMOS fast slew rate standard of 12 mA that can be seen as a source with 50Ω impedance. Therefore the CCLK line should be treated as a 50Ω transmission line and care should be taken not to branch the track or use a star topology. The end of the CCLK transmission line should be terminated with a parallel termination of 100Ω to VCC, and 100Ω to GND (the Thevenin equivalent of  $V_{TT} = V_{cc} / 2$ , and assuming a trace characteristic impedance of 50Ω).

#### ***General Layout***

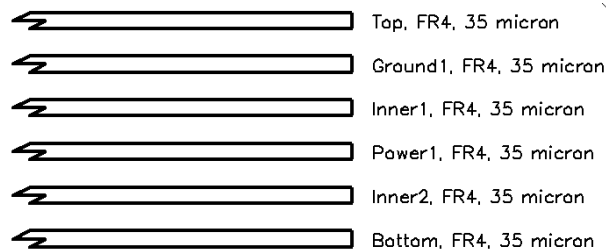
Because of the configuration interface high speed, special care was taken with the tracks for this interface by keeping them as straight as possible with the minimum vias used. The clock line use no vias as they introduce additional inductance. Figure 31 shows the layout of the clock trace including the stub for the termination resistors.



**Figure 31. Configuration clock board layout [33:34]**

The board thickness was made as thin as possible (1.55mm). This will increase the power plane decoupling capacitances and decrease spread inductance of the plane. Shorter length vias has lower inductance thereby increasing the signal integrity. On high-speed critical traces, such as the configuration interface and clocks, multiple vias was used, to further decrease inductance.

The layer stack-up used is shown in Figure 32. The critical signals were placed next to the power and ground planes to improve noise reduction.



**Figure 32. PCB Layer Stackup**

Although more screening can be achieved by burying the signal traces, it is offset by the fact that the component leads are the main parasitic antennas. Since the components will radiate anyway, burying the traces may not help that much.

### ***HyperLynx Simulation***

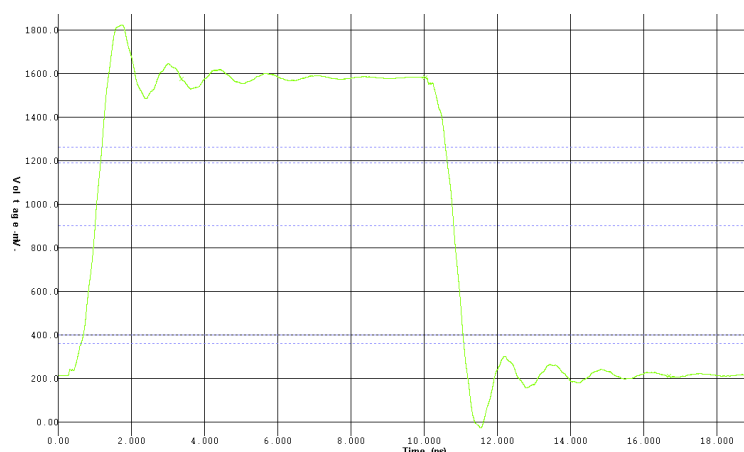
Only critical signals were simulated. The simulations were used to look at signal integrity. As seen before, the configuration I/Os on the FPGA use the LVCMOS standard. All the other I/O ports on the FPGA also use this standard. Table 17 shows the voltage range for the LVCMOS standard.

Standard	+3.3V			+2.5V			+1.8V			+1.5V		
	Min	Typ	Max	Min	Typ	Max	Min	Typ	Max	Min	Typ	Max
V <sub>CCO</sub> [V]	3.0	3.3	3.6	2.3	2.5	2.7	1.7	1.8	1.9	1.43	1.5	1.57
V <sub>IH</sub> [V]	2.0	–	3.6	1.7	–	2.7	1.19	–	1.95	1.05	–	1.65
V <sub>IL</sub> [V]	–0.5	–	0.8	–0.5	–	0.7	–0.5	–	0.4	–0.5	–	0.3
V <sub>OH</sub> [V]	2.6	–	–	1.9	–	–	1.3	–	–	–	1.05	–
V <sub>OL</sub> [V]	–	–	0.4	–	–	0.4	–	–	0.4	–	–	0.4
I <sub>IN</sub> [μA]	–	± 5	–	–	± 5	–	–	± 5	–	–	± 5	–

**Table 17. IO standards DC Voltage Specifications at Various Voltage References [30:258]**

When using the configuration prom in parallel mode the maximum allowable clock period is 30ns (see [29:31]) thus the maximum clock frequency is 33.33MHz.

Figure 33 shows the reconfiguration clock measured at the configuration PROM with R29 and R30 equal to 100 ohms. The overshoot is within the limits of the FPGA and configuration PROM.

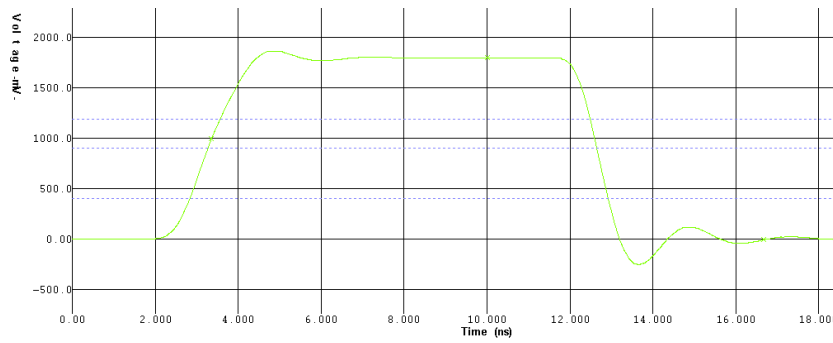


F = 50MHz      Overshoot 240mV      Undershoot 243mV

**Figure 33. Reconfiguration clock measured at configuration prom**

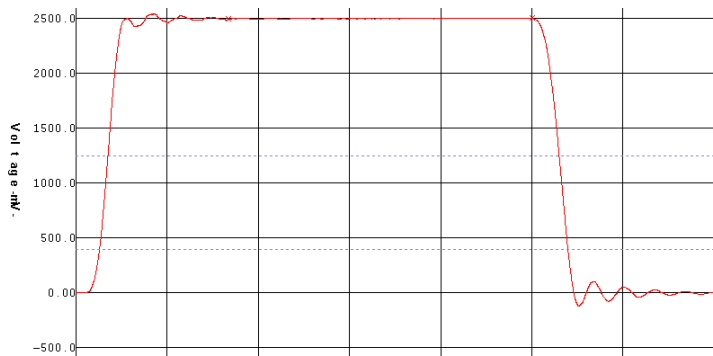
Figure 34 shows trace D1 of the SelectMAP configuration interface. This trace represents the longest trace in the configuration interface.



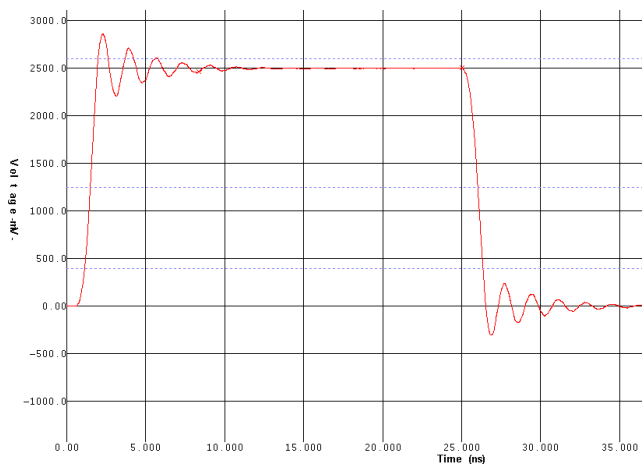


**Figure 34. Configuration PROM SelectMAP line D1**

Figure 35 shows the simulated waveform for the instruction bus line D8 measured at the Flash. This track is the longest track in the instruction bus. The FPGA is driving the bus at slow slew rate and 8 mA 2.5V 20MHz. Figure 36 shows the same waveform but the FPGA is now driving at 12mA. With the higher current, ringing is evident on the rising and falling edges.

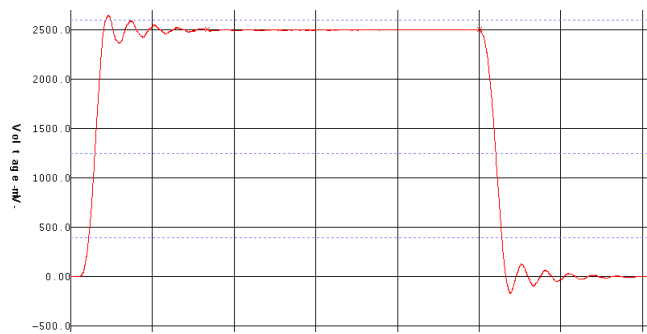


**Figure 35. Instruction bus line D8 at 8mA**

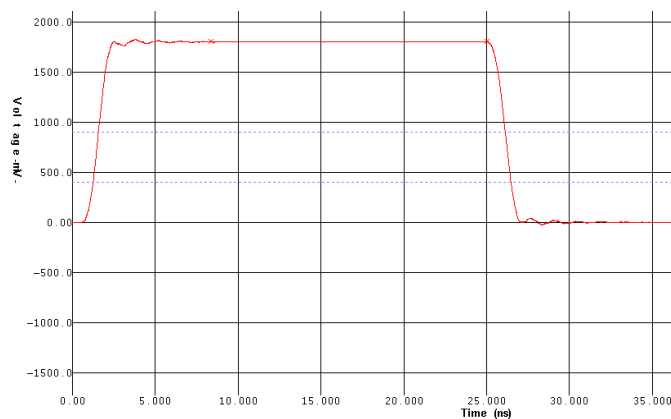


**Figure 36. Instruction bus line D8 at 12mA**

Figure 37 shows the A7 trace of the instruction bus measured at the Flash device. The FPGA is driving the bus at slow slew rate and 12 mA 2.5V 20MHz. Figure 38 shows the same waveform but with the FPGA driving at 8mA. Once again, it is evident that better results are obtained when the FPGA is driving the flash interface at 8mA.

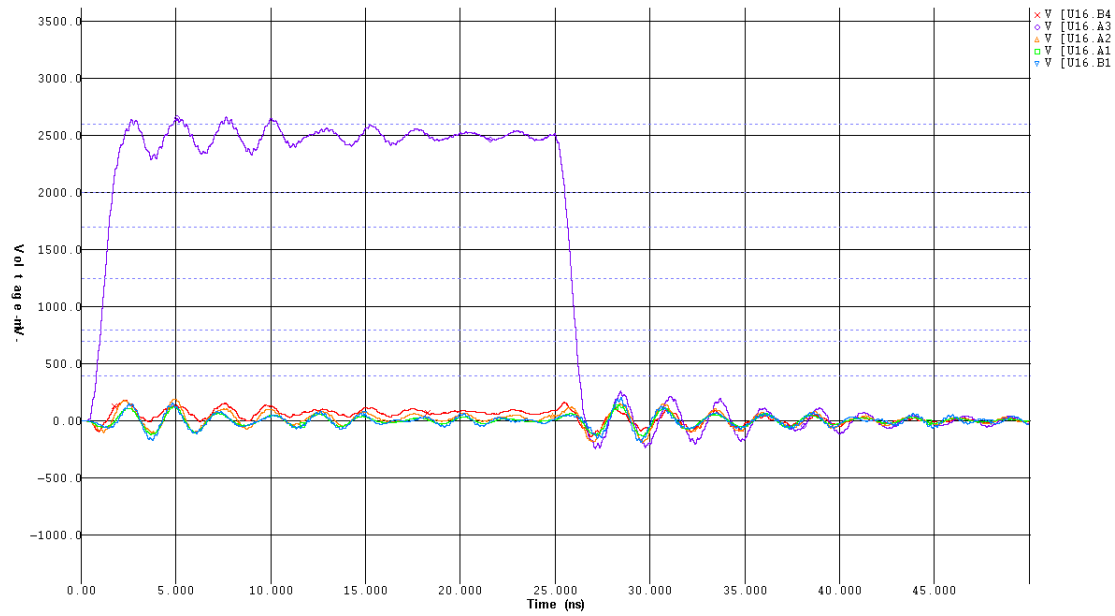


**Figure 37. Instruction bus line A7 at 12mA**



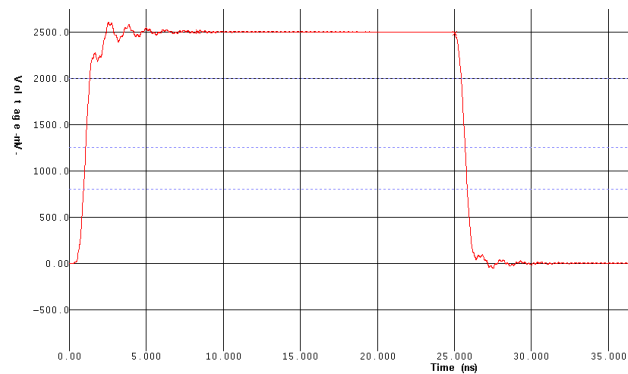
**Figure 38. Instruction bus trace A7 at 8mA**

Figure 39 shows crosstalk between adjacent tracks on the instruction bus at 20MHz. The FPGA is driving one address line at fast slew 8 mA. Increasing the current to 12 mA has a small impact on the cross talk. A3 (purple) on U16 is the source line here.

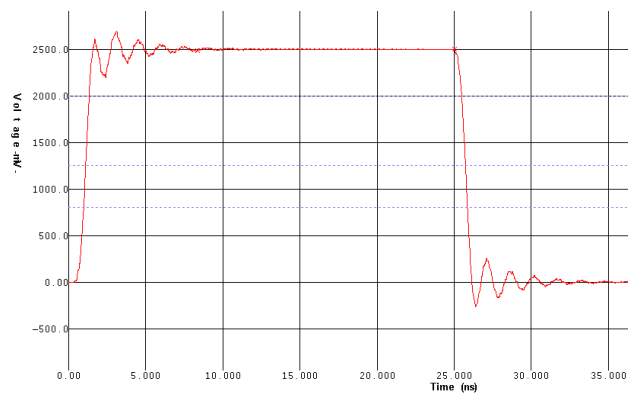


**Figure 39. Instruction bus cross talk**

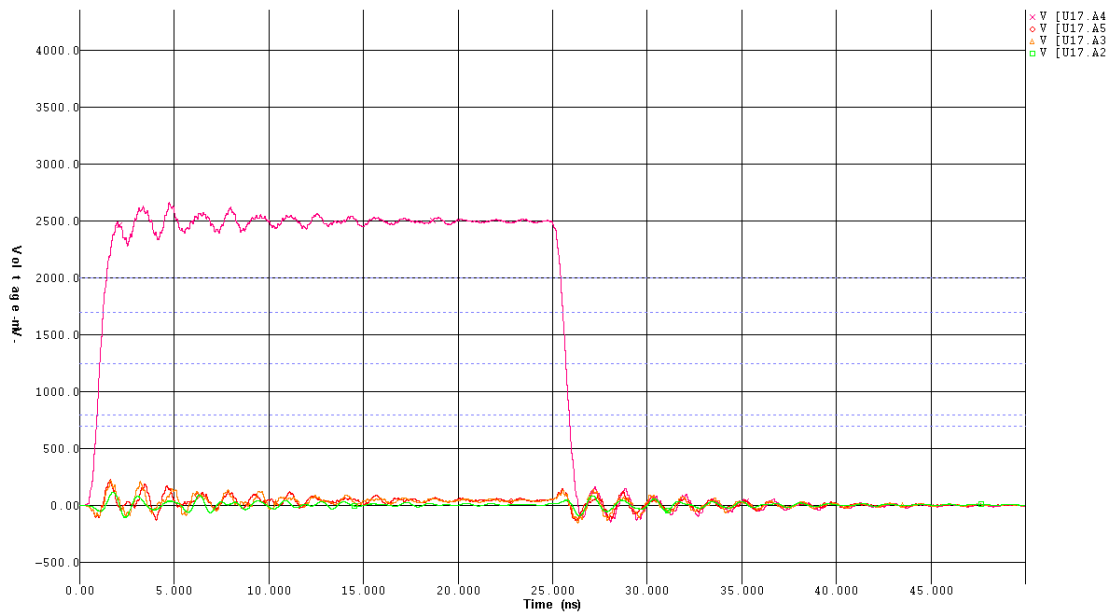
Figure 40 shows the A7 trace of the data bus while the FPGA is driving it at a fast slew rate 8mA 20MHz. Figure 41 shows the same waveform but at 12mA. Current limiting at 8mA is the better option.



**Figure 40. Data bus trace A7 at 8mA**



**Figure 41. Data bus trace A7 at 12mA**



**Figure 42. Data bus cross talk**

Figure 42 shows the cross talk on the data bus. The FPGA is driving A4 at fast slew rate 8 mA. The pink wave is the applied signal at the RAM chip. The other signals are from the adjacent tracks measured at the RAM chip. The cross talk is low as can be seen from this figure.

The HyperLynx simulation shows no signal integrity problems or any over voltage conditions. It is important to ensure that the over and under shoot of the signals are within the values in Table 17. Violating the specifications will not only damage the component, but will also cause higher power consumption due to the protection diodes conducting. If the maximum current is not exceeded, the device will not be damaged but power consumption and the core temperature of the FPGA will increase thereby increasing the leakage current.

## **Section 4.1.7 Software**

### **Section 4.1.7.1 Debugger**

Debugging software running on an embedded processor is difficult without development tools. Most commercial micro controllers come with some form of a debugging environment. In some cases the environment is free but often one has to buy the environment. When

choosing an embedded processor one of the main contributing factors is the debug environment.

In this research the soft-core implemented on the FPGA came without a debugger thus it was necessary to develop a debugger. Instead of reinventing the wheel the author looked at one of the most popular environments in both the PC and embedded market namely the GNU tools. The soft-core used already makes use of the GCC compiler thus it makes sense looking at GDB.

This section describes the implementation of a minimal GDB server that runs on the soft-core. The debugging session is split into two halves: a server, also called a GDB stub, that runs on the target being debugged and the debugger that runs on a host machine. The goal is to run as little code as possible on the chip. Anything that the remote machine can take care of gets isolated and runs on a remote machine. Symbol information, for example, is non-essential to debugging an application, so the remote machine instead of the stub can manage it. User interface is also a non-essential concern. The remote machine has the luxury of developing a grand user interface, but it makes less sense to have to run this code on the chip.

The LM32 core has an exception base address (EBA) and a debug exception base address (DEBA). When an exception occurs the CPU branches off to an address that is an offset from, either the EBA or the DEBA register. The DEBA register is used for the GDB stub to redirect all debug exceptions to the GDB stub exception handler. In debug mode the break point, instruction bus error, data bus error and watch point exceptions are handled by the GDB stub. All other exceptions are handled by the application being debugged.

Through the GDB stub the PC based debugger can interrogate the CPU by reading and writing to CPU registers and memory locations on the data and instruction bus. Hardware and software breakpoints can be set. The hardware breakpoint implementation is straight forward. Once the CPU receives the command and address for the break point it checks if one of the four hardware breakpoint registers (BP0 – BP3) are available and populates it with the address. The CPU hardware constantly compares the program counter (PC) with the BPx registers and when the PC match an exception is generated in which case the GDB stub exception handler takes over. The software breakpoint works similar except that there are no BPx registers and the CPU places a BREAK instruction at the address where the breakpoint

should occur. A copy is made of the replaced instruction which is restored once the breakpoint is cleared.

```

insn = *((unsigned *)registers[PC]);
opcode = insn & 0xfc000000;
if ((opcode == 0xe0000000) || // bi - Unconditional branch instruction
    (opcode == 0xf8000000)) // calli - 4 cycles
{
    branch_step = 1;
    //PC = PC + sign_extend(imm16 << 2) for bi
    //PC = PC + sign_extend(imm26 << 2) for calli
    branch_target = registers[PC] + (((signed)insn << 6) >> 4);
}
else if ( (opcode == 0x44000000) // be Branch if equal
|| (opcode == 0x48000000) // bg Branch if greater
|| (opcode == 0x4c000000) // bge Branch if greater or equal
|| (opcode == 0x50000000) // bgeu Branch if greater or equal, unsigned
|| (opcode == 0x54000000) // bgu Branch if greater, unsigned
|| (opcode == 0x5c000000) // bne Branch if not equal
)
{
    branch_step = 1;
    //PC = PC + sign_extend(imm16 << 2)
    branch_target = registers[PC] + (((signed)insn << 16) >> 14);
}
else if ( (opcode == 0xd8000000) // call
|| (opcode == 0xc0000000) // b Unconditional branch
)
{
    branch_step = 1;
    //Adds 4 to the PC, storing the result in ra, then unconditionally branches to the address in
    //rX (bits 25:21).
    branch_target = registers[(insn >> 21) & 0x1f];
}
else
    branch_step = 0;

/* Set breakpoint after instruction we're stepping */
seq_ptr = (unsigned *)registers[PC];
seq_ptr++;
seq_insn = *seq_ptr;
*seq_ptr = LM32_BREAK;
if (branch_step)
{
    /* Set breakpoint on branch target */
    branch_ptr = (unsigned *)branch_target;
    branch_insn = *branch_ptr;
    *branch_ptr = LM32_BREAK;
}
flush_i_cache ();

```

**Table 18. GDB-stub code segment for step command**

Implementing single step in the GDB-stub is more involved. Before stepping, the assembler instruction must be consider to determine what affect it will have on the program counter. Table 18 shows the implementation of the step command in C code.

Another useful debugging tool is the integration of the Xilinx JTAG programmer with the software development environment. This was accomplished through batch file processing. The Xilinx development environment contains a command line tool (data2mem) to map software binaries into the compiled bit stream without recompiling the bit stream. The FPGA user constraint file (UCF file) is used to label and specify the location of the affected BRAM regions (see Table 19). In Table 19 rom0, rom1, rom2 and rom3 combined forms a 2K by 32-bit emulated flash. Table 20 shows the configuration file for data2mem. Here we can see exactly how the binary address space maps to the BRAM inside the FPGA.

When the FPGA loads the modified bit stream at start-up the corresponding BRAM (flash emulation) regions are initialised with the binaries. To further speed up the process this new bit stream is immediately dumped through the Xilinx iMPACT load software (configuration file driven) into the SRAM of the FPGA after which the FPGA is reset. From compile time to code execution takes a few seconds and all of this is integrated into the make file of the Eclipse environment.

```

INST "rom0/ram_31_24" LOC = RAMB16_X0Y15;
INST "rom0/ram_23_16" LOC = RAMB16_X0Y14;
INST "rom0/ram_15_8" LOC = RAMB16_X0Y13;
INST "rom0/ram_7_0" LOC = RAMB16_X0Y12;

INST "rom1/ram_31_24" LOC = RAMB16_X1Y15;
INST "rom1/ram_23_16" LOC = RAMB16_X1Y14;
INST "rom1/ram_15_8" LOC = RAMB16_X1Y13;
INST "rom1/ram_7_0" LOC = RAMB16_X1Y12;

INST "rom2/ram_31_24" LOC = RAMB16_X2Y15;
INST "rom2/ram_23_16" LOC = RAMB16_X2Y14;
INST "rom2/ram_15_8" LOC = RAMB16_X2Y13;
INST "rom2/ram_7_0" LOC = RAMB16_X2Y12;

INST "rom_monitor/ram_31_24" LOC = RAMB16_X0Y11;
INST "rom_monitor/ram_23_16" LOC = RAMB16_X0Y10;
INST "rom_monitor/ram_15_8" LOC = RAMB16_X0Y9;
INST "rom_monitor/ram_7_0" LOC = RAMB16_X0Y8;

```

**Table 19. FPGA soft-core emulated flash storage description**

```

ADDRESS_SPACE lm32_ROM RAMB16 INDEX_ADDRESSING[0x00000000:0x00007fff]

    BUS_BLOCK
        rom0/ram_31_24 [31:24] PLACED = X0Y15;
        rom0/ram_23_16 [23:16] PLACED = X0Y14;
        rom0/ram_15_8  [15:8]  PLACED = X0Y13;
        rom0/ram_7_0   [7:0]   PLACED = X0Y12;
    END_BUS_BLOCK;

    BUS_BLOCK
        rom1/ram_31_24 [31:24] PLACED = X1Y15;
        rom1/ram_23_16 [23:16] PLACED = X1Y14;
        rom1/ram_15_8  [15:8]  PLACED = X1Y13;
        rom1/ram_7_0   [7:0]   PLACED = X1Y12;
    END_BUS_BLOCK;

    BUS_BLOCK
        rom2/ram_31_24 [31:24] PLACED = X2Y15;
        rom2/ram_23_16 [23:16] PLACED = X2Y14;
        rom2/ram_15_8  [15:8]  PLACED = X2Y13;
        rom2/ram_7_0   [7:0]   PLACED = X2Y12;
    END_BUS_BLOCK;

    BUS_BLOCK
        rom_monitor/ram_31_24 [31:24] PLACED = X0Y11;
        rom_monitor/ram_23_16 [23:16] PLACED = X0Y10;
        rom_monitor/ram_15_8  [15:8]  PLACED = X0Y9;
        rom_monitor/ram_7_0   [7:0]   PLACED = X0Y8;
    END_BUS_BLOCK;

END_ADDRESS_SPACE;

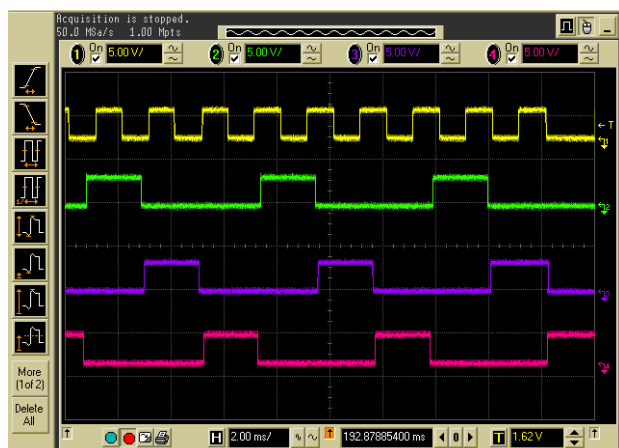
```

**Table 20. Configuration file for data2mem application**

### Section 4.1.7.2 OS support

Subsumption (see Section 2.2 ) relies heavily on parallel processing. To support this, the system makes use of a real-time multitasking operating system. After looking at various operating systems with a small footprint it was decided to use FreeRTOS. FreeRTOS is a lightweight pre-emptive operating system written in C. The core OS consists of only three C files. The porting to a new platform, like the LM32, is also fairly simple. In essence one has to write the code required for the OS timer tick, interrupt service routine, peripheral initialisation and processor start-up code. For this research, FreeRTOS V5.4.2 was ported to the LM32 processor with the timer tick rate set at 1kHz.





**Figure 43. FreeRTOS tasks**

To test the ported RTOS a small program was written which contained three tasks. Each task waits for a semaphore and once it is set the task toggles a I/O pin and then waits for the semaphore. When task1 completed one loop, it sets the semaphore for task 2. Task 2 and 3 executes in the same manner. Thus by using semaphores each task is depended on the others. All tasks have the same priority and will executed as follow: task1 -> task2 -> task3 -> task1 ..... Figure 43 shows the process. The yellow wave is generated at 1kHz by the operating system timer tick. The green wave is task1, purple is task 2 and red is task 3. From this we can see that the timer tick is always present and stable and the tasks executes correctly in sequence.

## **Section 4.2 Power supply**

One of the major problems when designing with a FPGA is inrush current. Xilinx claims that the Virtex 4 has no inrush current. Table 21 shows the current consumption at start-up. The values in Table 21 was measured with a power on sequence of  $V_{CCINT}$ ,  $V_{CCAUX}$  and lastly  $V_{CCO}$ . If power sequencing is implemented, this is the Xilinx recommended sequence.

It is very difficult estimating the power requirements for a FPGA design as it depends heavily on the HDL design on the FPGA. To overcome this challenge the PSU is designed with reserve capacity and adequate reservoir capacitors are placed on the board. At start-up the PSU is driving the loads as shown in Table 21 and a capacitive load consisting of all the decoupling capacitors on the board. To ensure a stable start-up the power supply contains a

soft start feature. According to the Virtex 4 documentation the ramp time for the supply voltage should not exceed 50 ms.

	Voltage [V]	VLX15 [mA]	VLX25 [mA]
$I_{CCINT}$	1.2	750	1350
$I_{CCO}$	3.3	75	100
$I_{CCAUX}$	2.5	100	125

**Table 21. Virtex 4 quiescent current**

Another common issue is power sequencing. The Virtex 4 is within limits immune to sequencing problems. With this in mind and the low start-up current, power sequencing will not be a design requirement for the power supply if all voltages will reach stability within 50ms after power up. The FPGA's internal POR circuit monitors the core and auxiliary supply. The auxiliary voltage ( $V_{CCAUX}$ ) should rise from 1 to 2.4V within 50ms, and the internal voltage ( $V_{CCINT}$ ) to 1 V.

For successful FPGA configuration and data retention the following voltage levels should be present at all times:

$$V_{CCINT} > 0.9V$$

$$V_{CCAUX} > 2V$$

$$V_{CC0} > 2V$$

$$V_{CC2} > 2V$$

$V_{CC0}$  &  $V_{CC2}$  are required for the FPGA's SelectMAP and configuration interface to work. For this design the configuration bank (0) and the SelectMAP bank (2) are connected to 2.5V. It is possible to connect these to 3.3V which would require a higher minimum voltage. A voltage supervisory circuit is used to ensure that the FPGA is always in a known and stable state at start-up. The core (1.2V), auxiliary (2.5V) and peripheral voltages (3.3V) are monitored. The reset output of the supervisory circuit drives the PROGRAM\_B input on the FPGA that acts as an asynchronous full chip reset. Table 22 shows the voltage threshold for the supervisory circuit. It is clear that they meet the requirements.

	Voltage [V]
$V_{CCINT}$	1.14
$V_{CCO}$	2.941
$V_{CCAUX}$	2.375

**Table 22. Reset voltage threshold**

The radio controlled market has an extensive range of servos. Most of these devices work with a 5V supply. For this reason the power supply will also provide 5V. The 5V supply does not require high load regulation and low noise and ripple levels.

Table 23 shows the maximum current requirements for each voltage. The design is battery driven and special care was taken to ensure maximum efficiency. A high frequency DC-DC converter with low dropout was used for the 1.2V, 2.5V, 3.3V and 5V supply. Due to the potential current consumption, linear regulators will create a substantial energy loss in the form of heat dissipation.

Voltage [V]	1.2	1.8	2.5	3.3	6
Max. Current [A]	2	0.1	2	3	2

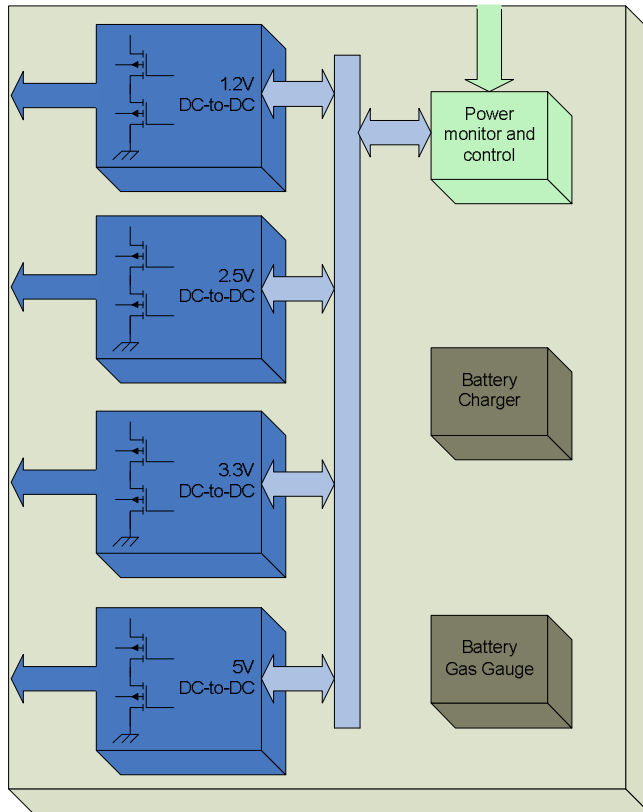
**Table 23. PSU maximum current design requirement**

Figure 44 shows the block diagram of the power supply.

The Xilinx parallel configuration PROM needs a 1.8V supply for its internal logic. According to the datasheet the PROM draws a maximum of 10 mA at 33MHz clock. The Virtex 4 can go up to 100MHz. Because of the low current consumption a linear regulator is justifiable. The smallest linear regulator supplies 100 mA which is more than adequate to drive the PROM at any clock speed.

The efficiency of a LDO regulator is limited by the quiescent current and input/output voltage as follows:

$$Efficiency = \frac{I_o V_o}{(I_o + I_q) V_i} \times 100$$



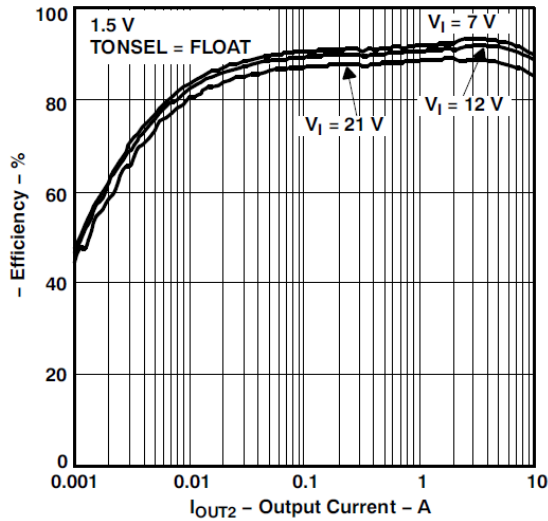
**Figure 44. PSU block diagram**

To have a high efficiency LDO regulator, the drop-out (voltage difference between input and output) and quiescent current must be small. For this purpose the Texas Instruments TPS73118 was used with an ultra low dropout voltage of 30mV and a quiescent current of 0.3  $\mu$ A. The 2.5V supply is used as the input voltage to the 1.8V LDO for the configuration PROM. To further decrease the voltage difference, an inline diode is placed at the regulator input decreasing the input voltage by a further 0.6V. This 1.8V LDO regulator is placed on the main board because it is only used by the configuration PROM.

### **Section 4.2.1 Component selection**

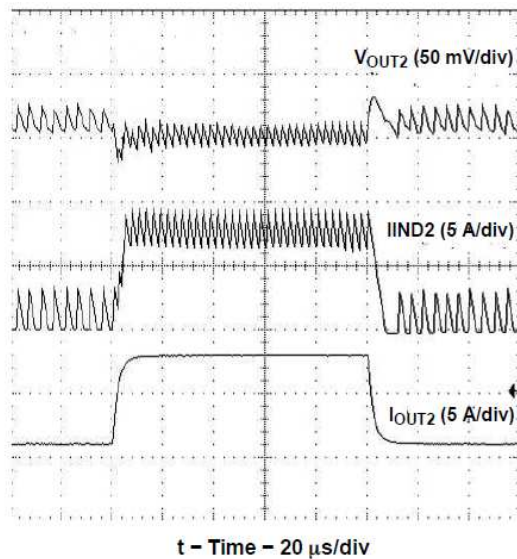
Board space and high efficiency was the driving force behind component selection.

A synchronous converter topology was used. For this the Texas Instruments dual synchronous controller is ideal. The TPS51124 incorporates variable frequency control to ensure high efficiency at light loads. The synchronous topology has the disadvantage of two external mosfets, but this is offset against increased efficiency compared to non-synchronous converters. Figure 45 shows the converter efficiency at different loads.



**Figure 45. Synchronous converter efficiency**

There is another aspect that makes the TPS51124 very attractive for this application. The FPGA is renowned for its high power supply transient's response requirements. Adequate decoupling capacitors plays an important role in supply transient response but it does not remove the requirement for a fast supply. Figure 46 shows the load transient response of the TPS51124.



**Figure 46. Converter load transient response**

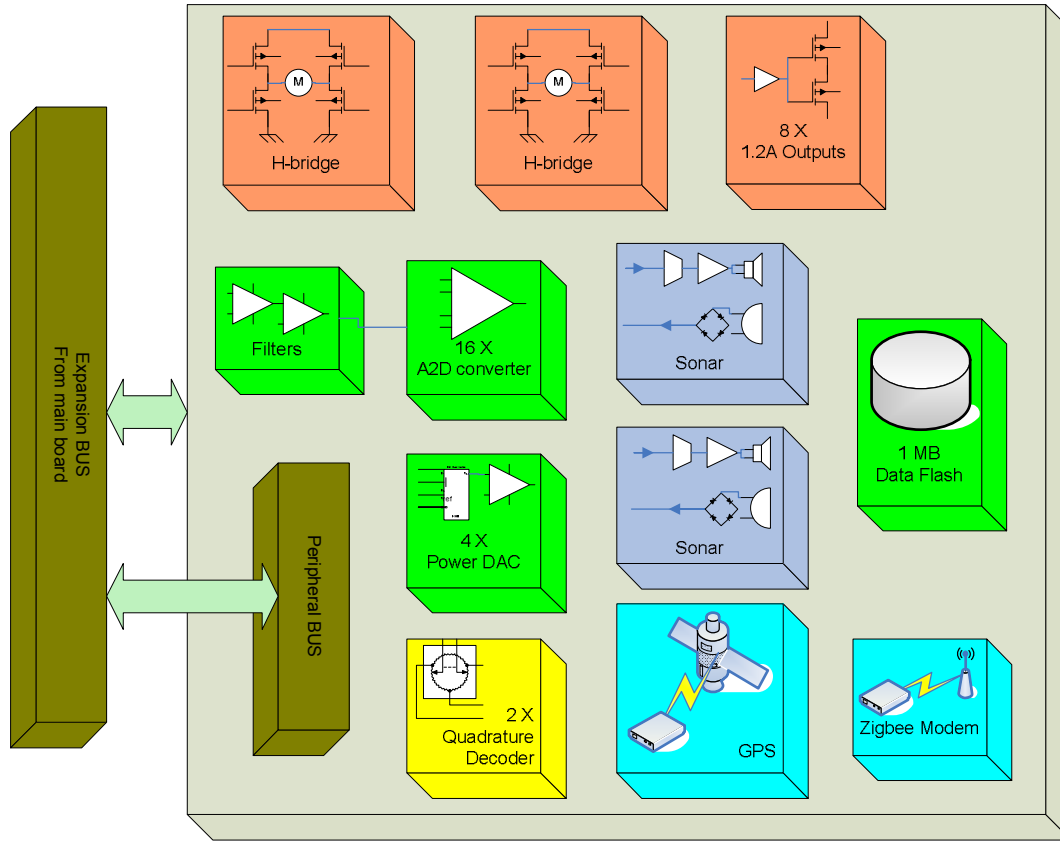
To navigate autonomously the robot must be battery driven. There are numerous battery technologies to choose from. The robot requires a small light weight battery pack with a high current density. The Lithium Polymer battery fulfils these requirements and with most mobile

devices using the Lithium Polymer batteries, it comes at a low price. There are however negative aspects with regards to this battery chemistry. Because lithium can easily catch fire when exposed to oxygen care must be taken when charging and discharging the battery pack. To manage the charging of the battery pack, a 2A Lithium Polymer switching charger design is included on the power supply board. The charger design is based on the BQ24103A from Texas Instruments.

A gas gauge IC is also included to measure the energy stored and withdrawn from the batteries. The gas gauge IC acts as an intelligent battery energy estimator, ensuring optimal battery usage. For this application the bq27210 from Texas Instruments was chosen. The system communicates with the gas gauge through an I2C interface. Through this interface the system has access to a host of information including battery voltage, battery temperature, relative state of charge, available capacity, compensated available capacity and many more. The device is a reduce pin component intended for single cell application. Instead of using the bigger footprint devices for multiple cell applications the bq27210 is fooled into thinking it is connected to one cell. This is done through a voltage divider that is amplified with an operational amp and connected to the battery voltage sense input of the bq27210.

### ***Section 4.3 Peripheral Board***

The main board does not make adequate provision for motor control and analogue inputs and outputs. It was decided to move all analogue and motor control circuits onto a separate board that mates with the main board through the expansion bus. The peripheral board block diagram is shown in Figure 47.



**Figure 47. Peripheral board block diagram**

The dual H-bridge controllers will drive the robots two DC motors connected to the wheels. The additional high current outputs can drive smaller motors, servos or infrared sensors. Robot wheel rotation speed and direction can be sensed through the dual quadrature encoder interfaces. Provision for a GPS receiver is made as an optional aid for navigation. Each robotic agent can talk to others in the swarm through the Zigbee mesh network. The filters, analog-to-digital (A2D) and digital-to-analog (D2A) components are used to interface to various sensors. The sonar circuits provide all the electronics to drive a sonar transducer. Sonar complements the infrared and shaft encoder sensors, and improves the sensing distance and accuracy.

### Section 4.3.1 Component selection

#### *Analog-to-digital converter*

The maximum theoretical signal-to-noise ratio for an A2D converter can be calculated by [38]:

$$SNR_{MAX} = 6.02 \cdot N + 1.76 \quad [\text{dB}]$$

where N is the number of bits of the A2D.

In the ideal A/D converter transfer function, each code has a uniform width. That is, the difference in analog input voltage is constant from one code transition point to the next. Differential nonlinearity, or DNL, specifies the deviation of any code in the transfer function from an ideal code width of 1 LSB.

Integral nonlinearity, or INL, is a result of cumulative differential nonlinearity (DNL) errors, and specifies how much the overall transfer function deviates from a linear response. INL is sometimes simply referred to as the linearity of the converter. The INL specification tells the designer the best accuracy that the A/D converter will provide after calibrating the system for gain and offset.

The spurious free dynamic range, or SFDR, is the ratio of the level of the input signal to the level of the largest distortion component in the FFT spectrum. This specification is important because it determines the minimum signal level that can be distinguished from distortion components.

Part #	Res [bit]	SINAD [dB]	MAX SNR [dB]	DNL [LSB]	INL [LSB]	SFDR [dB]	ENOB	Offset error [LSB]	Gain Error [LSB]	P @3.3V [mW]	Samples [KSPS]
ADS8345NB	16	85	98.08		±6	98	13.83			7.5	100
ADS8344E	16	84	98.08		±8	92	13.66	±1	±1	7.5	100
TLC3548	14	80.8	86.04	±0.5	±0.5	97	13.13			9.3	200
ADS7871	14		86.04	±0.5	±2		-0.29	±1		8.5	48
AD7888	12		74	±2			-0.29	±6	±2	3.5	125
ADS7951	12	71.3	74	±1	±0.75	84	11.55	±2	±1	6	1000
TLV2548	12	70	74	±1	±1	84	11.34	±2.5		5	200
MCP3304-B	13	78	80.02	±1	±0.5	92	12.66	±3	±1	1	100
AD7927BRU	12	70	74	±1.5	±1	78	11.34	±8	±1.5	3.6	200
MCP3208	12	72	74	±1	±1	86	11.67	±1.25	±1.25	1.1	100

**Table 24. A2D comparison**



The effective number of usable bits is calculated with the following equation (see [38]):

$$ENOB = \frac{SINAD - 1.76dB}{6.02} \quad [\text{bits}]$$

The entries in Table 24 are sorted according to price with the most expensive parts at the top. Price, power consumption, SINAD and ENOB was used as a metrics for choosing the A2D converter. The MCP3304 proved to be a very good choice especially when considering its price.

### ***Op-amp***

The OPA2350UA was chosen as the preferred op-amp due to its low noise specification ( $5nV/\sqrt{H_z}$ ) and rail-to-rail single supply operation. It is also pin compatible with the TL072 JFET low noise op-amp.

### ***Mosfets***

The IR mosfets was selected due to its ultra low on resistance and package size. The IRF5850 and IRF5852 contain two mosfets with reverse diodes in one 6-pin SOT23 package. The on resistance is between 0.09 and 0.135 ohm and the pulsed drain current is between 9 and 11 amps.

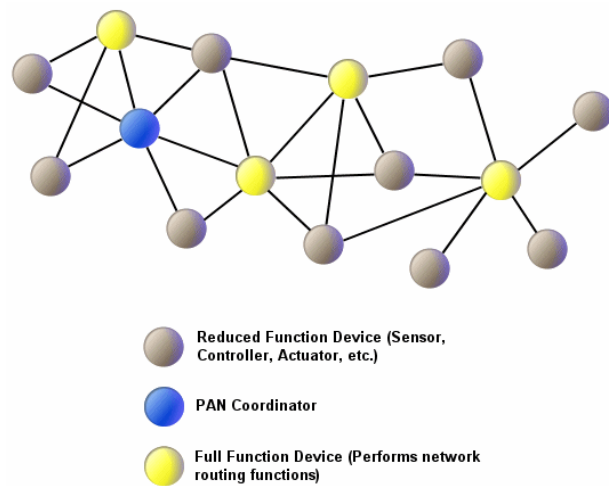
### ***GPS***

The Ublox LEA5-S was chosen as the GPS for this project. The device sensitivity is very high at 160 dBm which makes it suitable for indoor navigation. Ublox also includes their SuperSense technology which provides ultra-fast acquisition/reacquisition and exceptional tracking sensitivity.

Combining this with a passive 5 dBi antenna gives adequate indoor performance.

### ***Wireless modem***

The Zigbee technology was chosen due to the mesh network architecture and low power consumption. Through the mesh network robotic agents can talk to agents in close proximity or far away by relaying the message through other agents.



**Figure 48. Zigbee Mesh Network**

Two manufacturers were considered and Table 25 shows their products. The XBEE module was chosen above the ETRX one even though the ETRX draws less current. The reason behind this is the nature of the Zigbee standard. A Zigbee network is not homogeneous but consists of end points, a coordinator and a routers (see Figure 48). It is not recommended to make the coordinator and router battery operated since they must be on at all time. Thus only the endpoints are battery operated and can go into sleep mode. Another drawback is the size and overheads of the Zigbee standard, which lowers the effective data speed and in turn make longer use of the RF radio increasing the total power consumption.

Make and model	Sensitivity [dBm]	MAX. Tx current [mA]	MAX. Rx current [mA]	Transmit power [dBm]	Link Budget [dBm]	Max Distance [m]
XBEE	92	45	50	0	92	600
XBEE Pro	100	250	55	18	118	11400
ETRX2	99	41.5	37.5	5	104	2300
ETRX2-PA	97	130	37	18.5	115.5	8500

**Table 25. Zigbee modules**

The XBEE module overcomes these limitations by implementing a proprietary stack. Since interoperability is not required the XBEE proprietary stack (DigiMesh) is well suited for our application. The stack is smaller with less overheads thereby increasing bandwidth and

decreasing power consumption. DigiMesh has only one node type. All nodes can route data and they are interchangeable (see Figure 49). There is no parent-child relationship and all nodes can be configured as low powered devices.

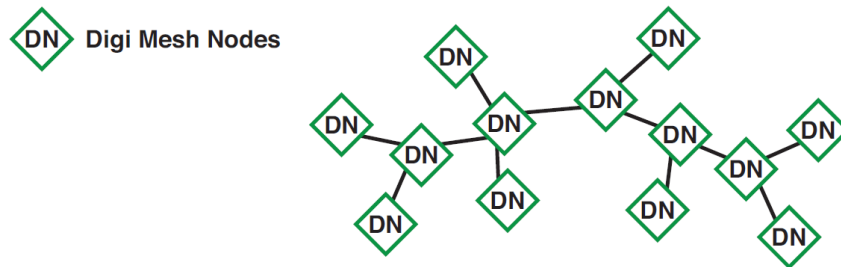


Figure 49. DigiMesh Network [39]

## Section 4.3.2 Design

### *H-Bridge*

The H-bridge design was kept simple due to the low voltage and current requirements. The typical drive motor used is between 6V and 12V with a stall current of 0.5A to 4A. Each H-bridge has a 4-pin connector containing the supply voltage to the H-bridge and the motor output. The output contains a multilayer varistor to clamp down on voltage spikes when driving an inductive motor load and the supply is protected with a TVS. Each mosfet is controlled individually and can be switch hard on and off through the 1.2A mosfet drivers. Figure 50 shows the H-bridge block diagram.

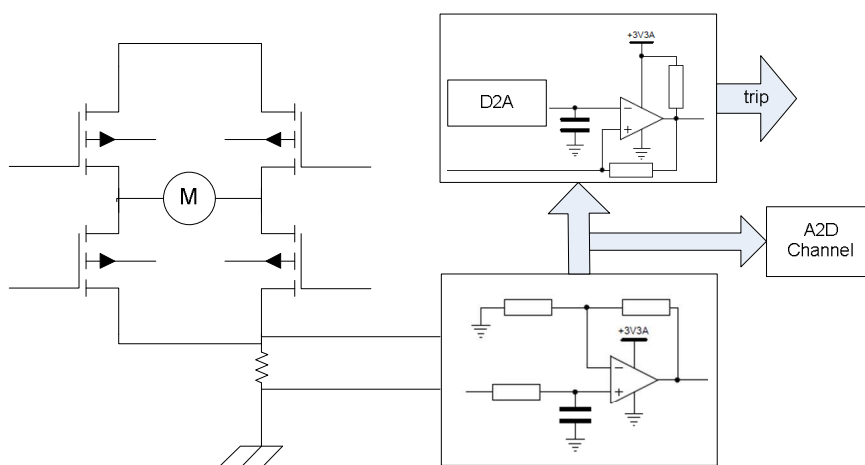


Figure 50. H-Bridge block diagram

The return line contains a sensing resistor which is read by one of the A2D channels through a low pass filter. The output of the filter also goes to a programmable comparator used to set the trip current for the H-bridge.

### ***Sonar***

Two sonar circuits are included on the board. In essence the circuit is the same as the one used on many other robots with a few exceptions. The sonar receiver circuit output is taken to an A2D converter for optional DSP processing. A programmable comparator is also included through which the sonar threshold detection can be changed dynamically. The threshold detection is used to interrupt the processor, only when a valid signal is present. All sensor intelligence is moved into the FPGA where, either the processor can process it, or a dedicated HDL module can process it in parallel and update shared memory through which the user application can read the distance. The complete sonar design is discussed in detail in Section 4.4.2 .

### ***IMU support***

For accurate navigation an IMU unit is essential. The development of such a module, though not very difficult, lies outside the scope and timeframe of this project. Therefore adequate provision for the integration of such a module is made on the peripheral board. An IMU usually contains a 3-axis accelerometer, a 3-axis gyroscope, temperature sensor, barometric pressure sensor and possibly also a 3-axis magneto-resistive sensors. Some of the latest components contains a digital interface although most of the devices on the IMU are analog. This requires numerous A2D channels. The peripheral board contains two A2D converters with 8 multiplexed channels each and the SPI and I2C bus is also made available on the expansion slot.

The difficult part with regards to the IMU is the software module/library which is usually resource intensive. For this reason there are enough RAM and ROM storage on the main board and the 32-bit RISC soft core can scale its frequency to 100MHz.

### ***Peripheral expansion bus***

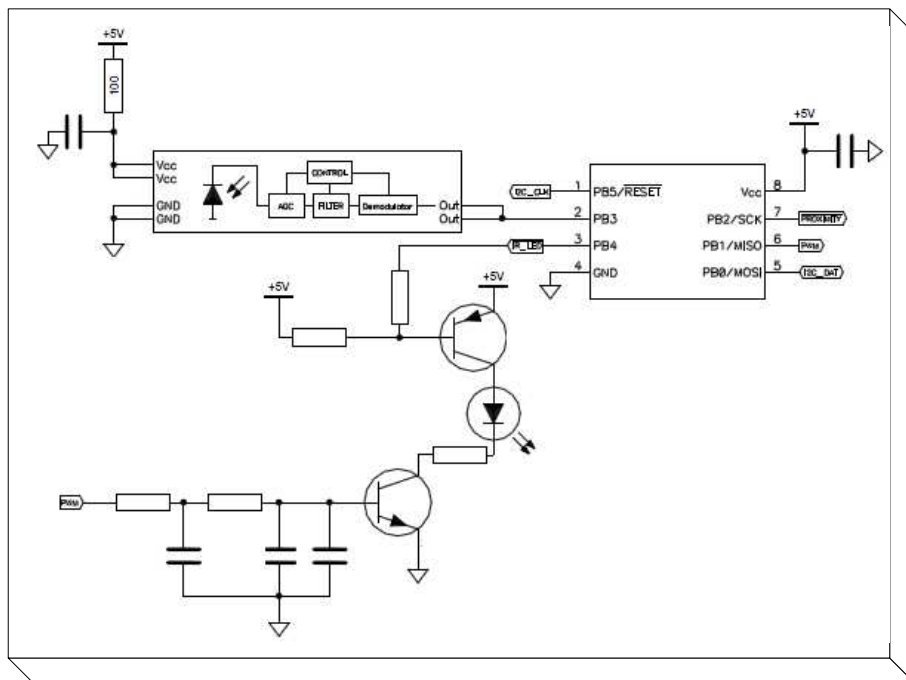
The main board peripheral bus expansion slot is duplicated through the peripheral board to facilitate the addition of hardware modules.

## Section 4.4 Sensors

### Section 4.4.1 Proximity

In the subsumption architecture the lowest behaviour is safe wandering (see Section 2.2.1.1). This behaviour is tasked with obstacle avoidance. The robot must be aware of obstacles in close proximity and far away. The objects further away are handled by higher behaviour responsible for path planning and map drawing, while the objects nearby is shared with the low safe wandering behaviour. To ensure robust behaviour the near and far object sensing must overlap to ensure sensor fall back if one of the two fails. To further enhance reliability a third sensor zone is created very close to the robot. This sensor zone acts like an emergency proximity sensor used as a last resort to prevent collisions.

Most small robot designers make use of an array of micro switches with whiskers for the emergency proximity detection. Some also use a flexible material in front of the micro switches to extend the sensing width of the micro switch (see [39]). It was decided not to go this route, but rather design a new technology.

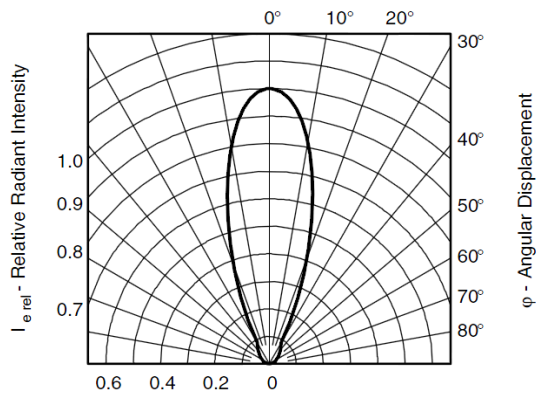


**Figure 51. Proximity smart sensor schematic**

The proximity sensor put forward in this research was designed from the ground up specifically for this project. The design is based on infra red reflection and makes use of a

tiny microcontroller to create an intelligent sensor network for proximity detection. Initially the small SOT-23 PIC10F was used, but its processing power is inadequate to do the modulation, demodulation, PWM generation and it cannot meet the timing constraints. The PIC10F has a 1:4 MIPS to MHz ratio with a maximum clock speed of 4MHz and does not include a PWM generator. For a second iteration the AVR ATtiny45 was chosen with a 1:1 MIPS to MHz ratio and a maximum clock speed of 10MHz. The ATTiny has more RAM, flash and I/O pins compared to the PIC and also includes a PWM generator. Figure 51 shows the design for the proximity sensor.

The Osram SFH5110-38 was used as the infrared detector and demodulator. The modulation frequency is 38kHz. The SFH5110 contains the infrared receiver, preamplifier, automatic gain control, band pass filter, demodulator and optical daylight filter. The daylight filter is optimised for 950nm wave length. To achieve maximum efficiency and distance, the infrared transmitter centre frequency should be as close as possible to 950nm. A 940nm infrared LED was used as the transmitter. For proximity detection it is important to choose a LED with a narrow beam to reduce false sensing due to wide reflections. The TSAL6200 is a 940nm infrared LED with a narrow beam (see Figure 52).



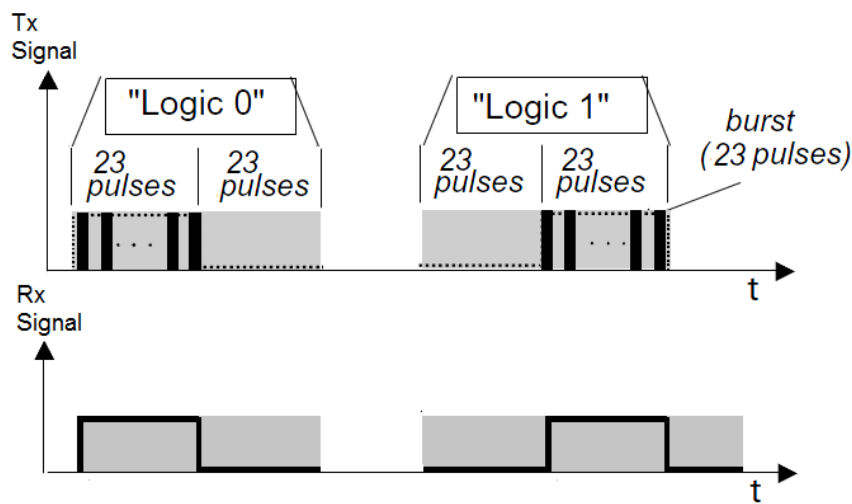
**Figure 52. TSAL6200 Infrared LED beam width**

The SFH5110 has a low power supply rejection ratio and the supply RC-filter was added to lower the noise. This proved to be very effective and a reduction in detector output noise was noticed.

The microcontroller modulates the signal at 38kHz to the IR LED. A digital to analog converter is created through the PWM port on the micro controller which goes through an active low pass filter. At first an op amp was used, but after some experimentation a

transistorised active filter performed similar. The digital to analog voltage is use to control the transmitting power of the IR LED. This is necessary to dynamically control the sensing distance to compensate for different ambient light and transition from indoor to full sunlight outdoor environment.

The typical sensitivity of the SFH511X family is specified for burst lengths of 600  $\mu$ s. Using a carrier frequency of 38kHz, this corresponds to 23 single pulses ( $605\mu\text{s} = 23 \text{ pulses} * 1/38 \text{ kHz}$ ). The transmitter protocol utilises a fixed length bit word for logic 0 and logic 1. Each bit word is sliced in two opposite halves (see Figure 53).



**Figure 53. Fixed length bit word for logic 0 and 1**

From Figure 53 it is evident that a logic 0 at the receiver after demodulation is seen as a logic 1 followed by a logic 0 pattern each with a width of 605 $\mu$ s. A Logic 1 is seen as a logic 0 followed by a logic 1 patter.

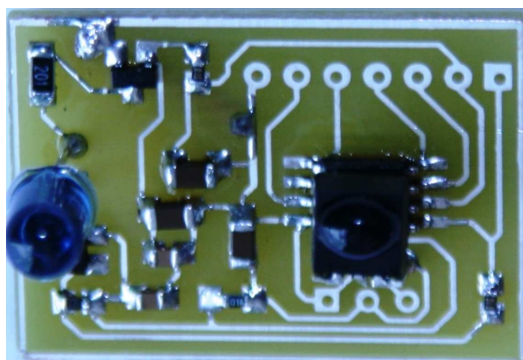
Unfortunately both the sun and some indoor lights radiate infra red which creates excessive noise at the detector output. It is not adequate to do basic modulation. A more intelligent signal processing technique is required for reliable indoor and outdoor applications. Correlation was used to bridge this gap. Correlation is a mathematical operation that is very similar to convolution. Just as with convolution, correlation uses two signals to produce a third signal. This third signal is called the cross-correlation of the two input signals. If a signal is correlated with itself, the resulting signal is called the autocorrelation.

The proximity detection is made up of cycles. Each cycle begins with a start bit, a logic 0 bit word followed by a logic 1 bit word. The remaining part of the cycle is made up from a random generated 8-bit value. The value is sent using the logic 1 and 0 bit word. Each logic 0 and 1 is sampled 4 times at the receiver and a logic low and high counter is incremented accordingly. At the end of the bit transmission these counters are checked against a threshold to determine if a valid word pattern was received. If so the correlation threshold counter is incremented. If not the correlation counter is decremented. All eight bits in the cycle is send and sampled in this manner. Once this process is completed, the correlation counter is checked against a correlation threshold and if it is above the threshold the cycle was received successfully and does not represent noise or an erroneous cycle. To further increase the reliability of this sensor, consecutive random number cycles are transmitted and once 15 consecutive valid cycles was received, it represents a valid reflection and the state is set as proximity detected. To clear the proximity detected latch, 5 consecutive invalid cycles must be received while transmitting valid cycles. These values were not chosen arbitrary but came from experimentation in indoor and sunlight conditions.

The best response for a proximity detection is:  $15 \times (368 \text{ us} + 3612 \text{ us}) = 59.7 \text{ ms}$ .

Each sensor has an I2C port and an open collector interrupt line. Through the I2C bus a proximity sensor network is created around the robot. The interrupt line signals a proximity event. Before installing a sensor into the network it must be assigned an unique address.

The transmitted power of the IR LED is controlled by a command on the I2C bus. The robot has a light sensor installed which it uses to detect ambient light. Thus, when the robot moves from indoor to outdoor sunlight, it will detect it and increase the sensors transmitted power to compensate for the increased ambient IR light.

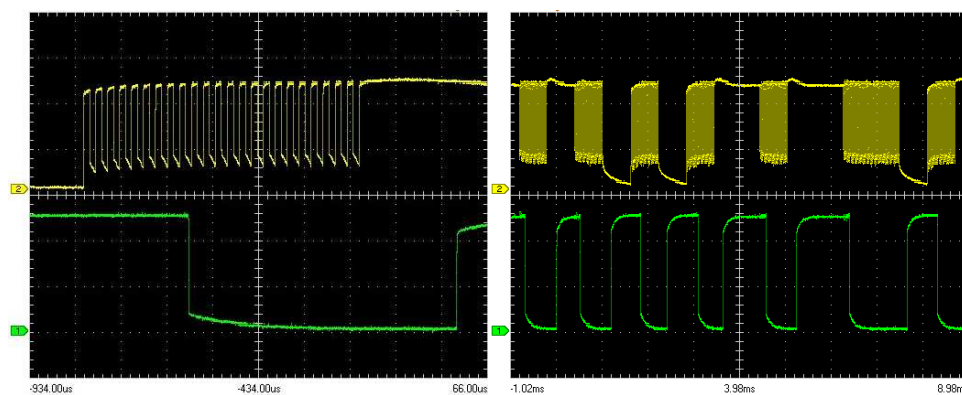


**Figure 54. Prototype circuit for proximity detector**



Figure 54 shows the prototype for this sensor. The small AVR micro controller is directly below the IR detector in the middle of the board. It was necessary to put rubber tubing over the transmitting LED to ensure that there is no crosstalk with the detector. Without this tube the noise at the receiver increases dramatically. The connector exposes the power pins and communication pins.

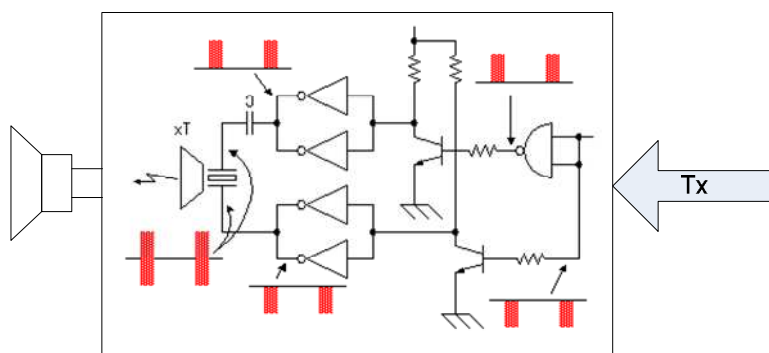
Figure 55 shows the waveforms for a detected object. Channel 2 (yellow) shows the modulated transmitted signal. Channel 1 (green) shows the demodulated received signal which is taken to the microcontroller for analysis.



**Figure 55. Proximity sensor object detected**

### Section 4.4.2 Sonar

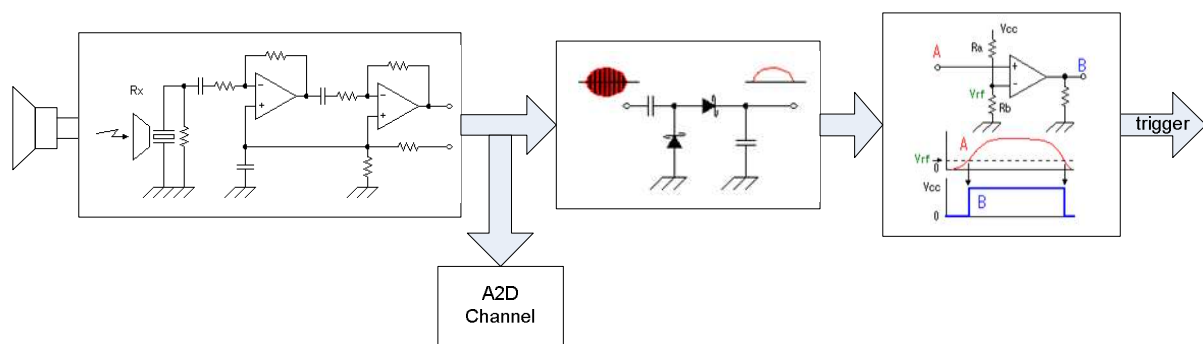
A CMOS inverter is used to drive the ultrasonic transmitter (see Figure 56). Two inverters are connected in parallel because of the transmission electric power increase. The phase of the voltage applied to the positive and negative terminal of the sensor is 180 degrees shifted.



**Figure 56. Sonar transmitter block diagram**

Because it is switching the current to the capacitor, about twice the voltage of the inverter output are applied to the ultrasonic transmitter. The microcontroller generates the modulation Tx stream.

The received signal is amplified with a single ended operational amplifier. A biased voltage is added to ensure bipolar operation. The amplified signal is taken to an analog to digital channel and to a detection circuit. The detection circuit is used to determine the presence of the ultra sonic signal. The received signal is passed through a Schottky half wave rectifier producing a DC voltage. This DC voltage from the detection circuit is monitored by a programmable comparator. The  $V_{rf}$  voltage is adjusted by a digital potentiometer connected to ground and 3.3V. The sonar software can adjust the threshold dynamically. Figure 57 shows the block diagram for the receiver circuit.



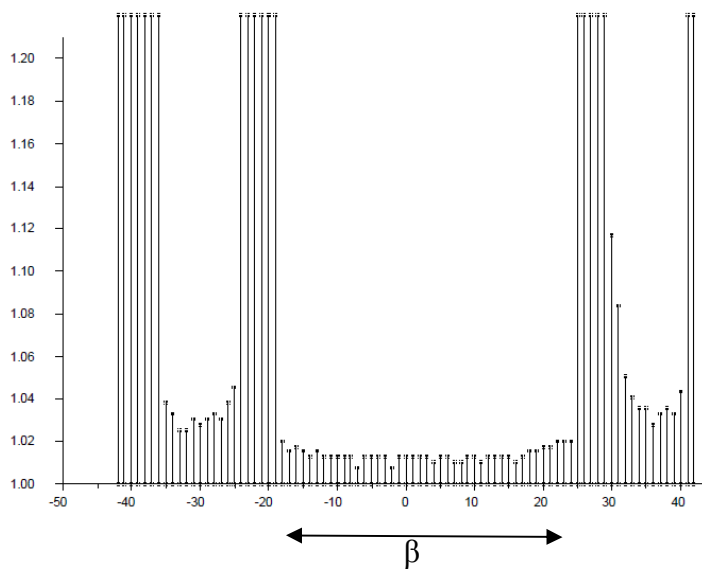
**Figure 57. Sonar receiver block diagram**

The ultrasonic receiver transducer is very sensitive to any sonic or electrical residual such as ringing of the transducer or noise created during the transmitted chirp. At close proximity the reflected signal is superimposed onto the induced noise. If one wants to sense at close proximity the threshold must be adjusted higher to ensure that the reflected signal trigger the detection and not the noise. At long distances this does not matter, as most robotic sonar's disable the receiver directly after transmission for a long enough period to miss the noise.

The sonar can be used as a simple proximity sensor with a longer range compared to the proximity sensor in Section 4.4.1. The only control required here is the setup of the threshold detector. The output of the threshold detector can serve as an interrupt to indicate object detection. By changing the threshold dynamically one can scan for near or far objects. Although this technique is simple and not processing intensive, it has serious drawbacks. It is

difficult if not impossible to determine the shape of the detected object (distinguish between plane, corner, cylinder and edge).

Direct sonar sensing for mobile robot navigation by John J Leonard [41] describes a more accurate method to determine the shape of the detected object. For this reason the sonar is mounted on a servo which rotates at predetermined fixed angle. This returns a reading range (see Figure 58) which is an ordered pair consisting of the range value  $r(k)$  and the sensor orientation  $\alpha(k)$ . The reading vector is defined as:  $\overline{r_i(k)} = (r_i(k), \alpha(k))$ .



**Figure 58. Typical wall response. Range vs. orientation plot [41:37]**

Sonar scans as shown in Figure 58, contains regions of constant depth (RCD) which are formed by strong returns. Weak sonar returns which do not form RCD can be explained by undesirable characteristics of the standard sonar hardware [41:38].

The range difference of the returns is defined as the absolute value of the difference between the minimum and maximum return. If  $\delta_R$  is the range difference threshold, then a RCD is defined as a set of continues adjacent returns of which the range difference is smaller than  $\delta_R$ . The width  $\beta$  of an RCD is the difference in angle between the left and right most return of the RCD.

The uncertain process by which weak returns are produced means that if the sensor is slightly rotated in either direction, the range can change considerably. In contrast, the well-defined

nature in which strong echoes exceed the receiver threshold means that the range will not change significantly for small changes in sensor orientation. To distinguish between strong and weak returns a width threshold is defined  $\beta_{\min}$ . A strong return is defined as a RCD with  $\beta \geq \beta_{\min}$  while for a weak return  $\beta < \beta_{\min}$ . Figure 58 shows one RCD region of strong return and one of weak return. Typical values for  $\beta_{\min}$  are 5 to 10 degrees [41:42]. The order of a return is defined as the number of surfaces that the sound has reflected from before returning to the transducer. Orienting the transducer perpendicular to wall will produce a first order return. Corners produce second order returns. All returns in a RCD are the same order and so the order of the RCD is the order of its returns. Figure 58 shows a first order RCD.

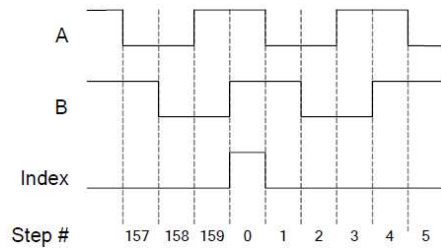
### Section 4.4.3 Shaft Encoder

Accurate determination of distance travelled and turn rate is essential especially when this data is used for dead reckoning. Dead reckoning is the process of estimating the robot's current position based on previous determined position. A disadvantage of dead reckoning is that since the positions are calculated based on the previous position, the errors of the process are cumulative thus the error grows over time.

There are an extensive range of shaft encoders on the market but most if not all of them are expensive. The author could not find a small shaft encoder with at least 1024 positions at a reasonable price. This created the need to design an encoder for this specific application.

The design is based on the AS5306A/5304A IC from Austria Microsystems. A 72-pole ring magnet is used which mounts on the robot axel stub and rotates past the AS5306. The AS5306 contains 4 Hall Effect sensors each placed in a quadrant on a 2 dimensional Cartesian plane and a DSP processor. The alignment of the ring magnet to the AS5306 must be within 1mm to achieve good accuracy.

The AS5306 outputs 40 quadrature pulses and 1 index pulse per magnetic pole pair. For this application a 72 pole (36 pole pairs) magnetic ring is used which translates to 1440 quadrature pulses and 36 index pulses per rotation.

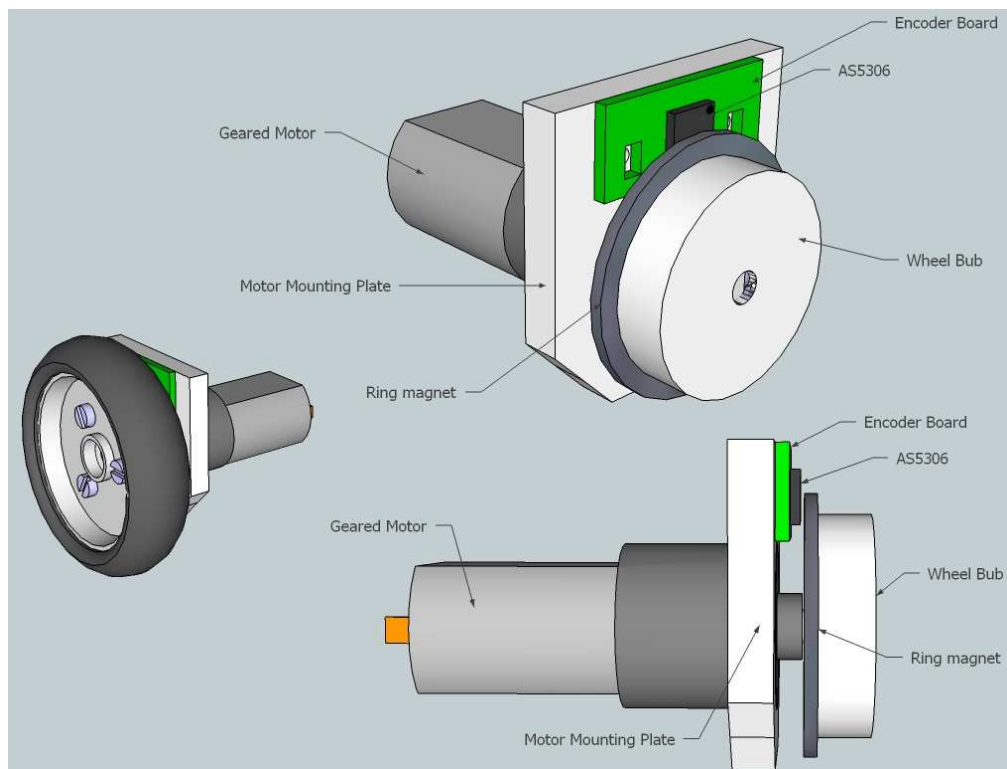


**Figure 59. Quadrature Decoder steps**

The 40 quadrature pulses translate to 160 steps as seen in Figure 59. Thus the maximum resolution for the encoder is:

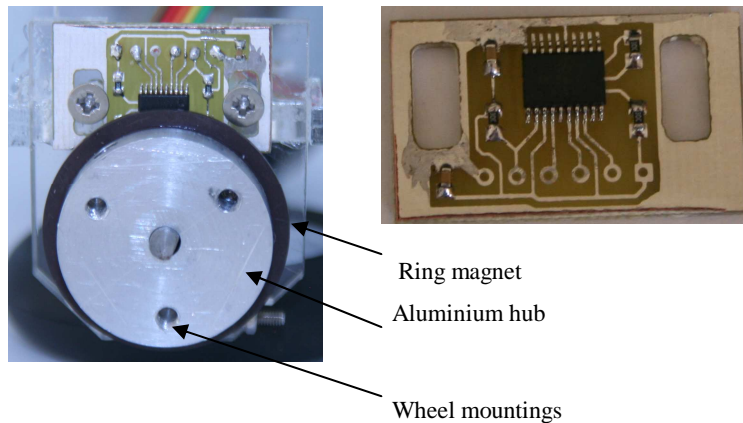
$$\begin{aligned}
 \text{Resolution} &= [\text{interpolation factor}] \times [\text{number of pole pairs}] \\
 &= 160 \times 36 \\
 &= 5760 \text{ steps per revolution} \\
 &= 0.0625^\circ \text{ per step}
 \end{aligned}$$

$$\begin{aligned}
 \text{Maximum rotation speed} &= 300\,000 / [\text{number of pole pairs}] \\
 &= 8333 \text{ rpm}
 \end{aligned}$$



**Figure 60. Hub assembly showing shaft encoder**

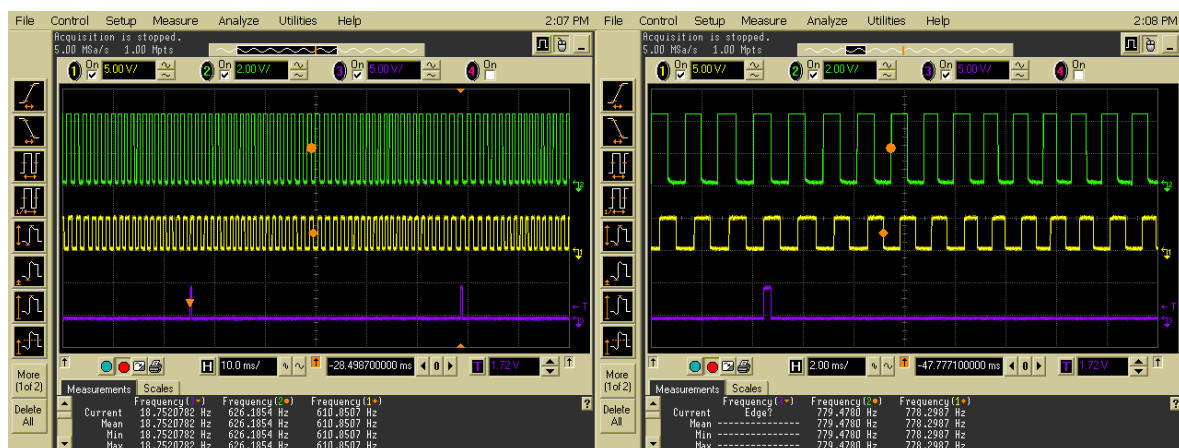
### Section 4.4.3.1 The circuit



**Figure 61. Shaft Encoder PCB and installation**

Figure 61 shows the assembled shaft encoder board used on each of the two robotic wheels. It is very important that the magnetic ring is perfectly centred on the motor shaft to avoid pulse jitter at the output of the encoder. The encode PCB is properly aligned with the magnet through the slotted mounting holes. This is accomplished by placing an oscilloscope on the sensor output signals and adjusting the height to produce the lowest jitter.

Figure 62 shows index and position waveforms for the design. The green wave is phase-A, yellow phase-B and the purple wave is the index pulse.



**Figure 62. Shaft Encoder waveforms**

## **Section 4.5 Assembly and testing**

Assembly and debugging took considerably more time than anticipated. When working with an off-the-shelf CPU a lot is taken for granted. In a design like this there are numerous factors to take into consideration during testing. When faced with an issue, it is sometimes difficult to determine if it is software related or hardware, and if so, is it the soft processor core, a peripheral, possibly the bus or timing on the bus.

### **Section 4.5.1 Main board**

The main PCB was tested by the manufacturer before delivery. When the board was received all power supply and ground pins were tested for short circuit, connection and correct pin location. The JTAG interface was also tested to the FPGA and the configuration PROM. All polarised capacitors were tested for reverse polarity. Lastly the FPGA configuration interface was tested. All tests were conducted with a multi-meter and all traces under testing were also tested to ground and all supplied voltages for short circuits. The power-on-reset circuit was tested for continuity and device pin placement. After passing all of these tests the board was ready for manufacturing.



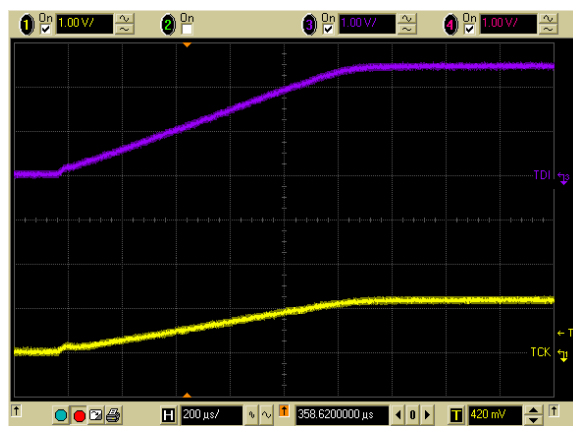
**Figure 63. Top and bottom view of the main board**

To reduce the cost and to simplify debugging only the BGA components (FPGA, configuration PROM, Flash and RAM) was placed by the contract manufacturer. Each BGA



was also X-rayed to check for short circuits, dry joints and bad solder joints. Figure 63 shows the assembled main board. All the components are not placed yet. Once debugging is completed the board will be fully assembled.

The rest of the components were placed by hand. At first only the essential components was soldered on the PCB, to power up the board and test for short circuits on the power supplies. The board was connected to a programmable supply with fine current limiting. After successful power-up more components were placed to test the JTAG interface and the serial configuration of the FPGA. Figure 64 shows the rise time of the 1.2V and 2.5V supply.



**Figure 64. Power supply rise time**

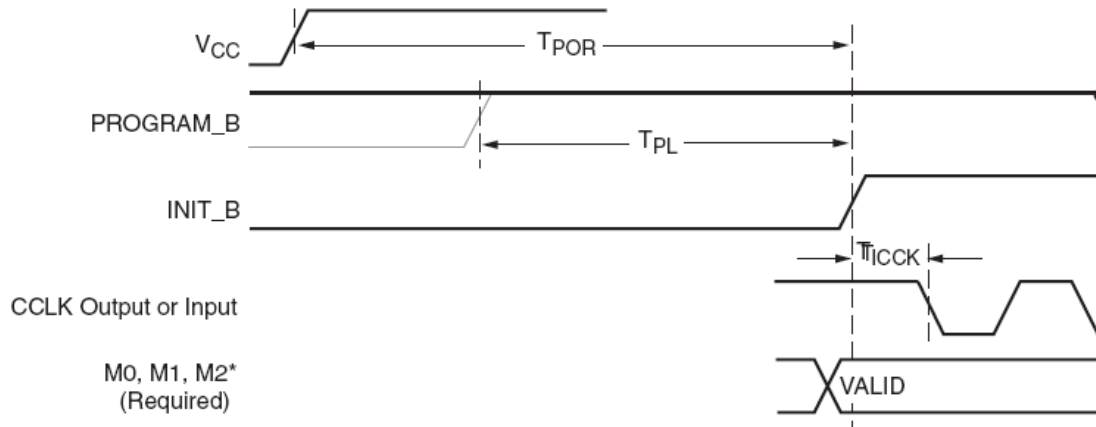
Symbol	Description	Ramp Time	Units
V <sub>CCINT</sub>	Internal supply voltage relative to GND	0.20 to 50.0	ms
V <sub>CCO</sub>	Output drivers supply voltage relative to GND	0.20 to 50.0	ms
V <sub>CCAUX</sub>	Auxiliary supply voltage relative to GND	0.20 to 50.0	ms

**Table 26. FPGA power supply ramp time [31:7]**

Table 26 shows the supply ramp time requirements for the FPGA. The configuration PROM has the same requirements [32:25]. From Figure 64 it can be seen that the power supply takes just over 1000  $\mu$ s for all the FPGA voltages to stabilise. This is compliant with the FPGA and configuration PROM requirements.

The next aspect to verify is whether the FPGA and the configuration PROM goes through a valid power-on-reset cycle.

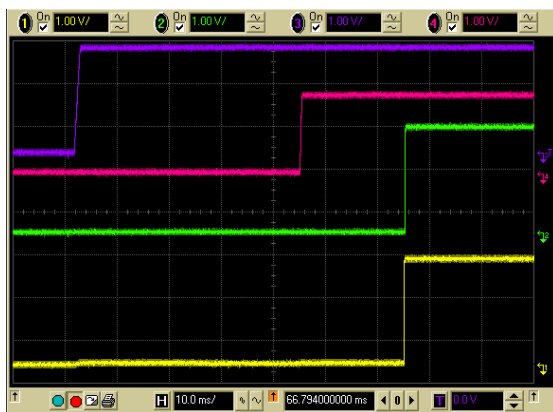




**Figure 65. FPGA power-up [33:16]**

On the FPGA the PROGRAM\_B pin functions as an asynchronous reset and when it is low the FPGA is held in reset state. INIT\_B reset the configuration PROM when it is low. If the power supply ramp rate is slower than 50ms the start-up must be delayed by keeping PROGRAM\_B low. The main board contains voltage supervisory circuit for each supply voltage. These are daisy chained with the PROGRAM\_B input of the FPGA. The 1.2V POR circuit drives the 2.5V which drives the 3.3V that finally drives the PROGRAM\_B input which ensure that the FPGA always start in a stable and known state. The power-on reset delay of each voltage can be set by changing C17, C18 and C22.

Figure 66 shows the POR waveforms and it is clear that the FPGA completes the POR and goes on to enable the configuration PROM (INIT\_B=high). To clock the data out of the configuration PROM PROGRAM\_B and INIT\_B must be high which is valid in Figure 66. The purple wave is the 2.5V valid signal, the green wave is the INIT\_B signal, the yellow wave is the PROGRAM\_B signal and the pink wave is the 1.8V valid signal.



**Figure 66. Power-on-reset**

Power-on-reset time ( $T_{POR}$ ) and program latency ( $T_{PL}$ ) shown in Figure 65 are defined as (see [34:35]):

$$\begin{aligned} T_{PL} &= 0.0005 * frames & [\text{ms}] \\ T_{POR} &= T_{PL} + 10 \\ frames &= \text{config. Frames} + \text{non-config. Frames} \end{aligned}$$

For the Virtex-4LX15:  $T_{PL} = 0.0005 * (440 + 6940) = 3.69\text{ms}$   
 $\therefore T_{POR} = 3.69 + 10 = 13.69\text{ms}$

The enable pin of the 1.8V regulator is driven by the 2.5V voltage supervisory circuit that in turn is driven by the 2.5V supply and the 1.2V voltage supervisory circuit.

The delay for each voltage supervisory circuit:  $Delay(s) = \frac{C_T(nF)}{175} + 0.5 * 10^{-3}$  [s]

For a 3.3nF capacitor: Delay = 19.36 ms

Thus the 1.8V is delayed by 38.71ms after the 2.5V. The 2.5V is the first active voltage and the 1.8V is the last available voltage. The voltage supervisory circuit continues to hold the PROGRAM\_B low (FPGA in reset) while it is waiting an additional 19.36ms for the 3.3V supply. Thus the power supervisory circuit holds the FPGA in reset for a total of 58.05ms. While it is waiting for the 3.3V the FPGA can complete its POR because it has a valid 1.2V and 2.5V. The FPGA takes  $T_{POR} = 13.69\text{ms}$  to complete its POR which is shorter than the final 19.36 ms that the voltage supervisory circuit is waiting for the 3.3V. Thus when the voltage supervisory circuit has releases the FPGA from its reset state, the FPGA can immediately start clearing its memory.

The configuration PROM keeps the INIT\_B low until  $T_{OER}$  seconds after the 1.8V is available.  $T_{OER}$  for the XCF08P is 0.5 to 30ms (p27 [29]). Thus, the configuration PROM will release the INIT\_B line

$$1.8\text{V PSU start-up time} + T_{OER} = 39.21 \text{ to } 68.71 \text{ ms}$$

after the 2.5V is valid. In Figure 66 the delay from a valid 2.5V supply to a high PROGRAM\_B and INIT\_B is about 58 ms that is within the calculated delay.

After the 58.05ms delayed start the FPGA and the configuration PROM has already completed their POR state and therefore the INIT\_B and PROGRM\_B can be driven high so that the configuration process can start.

The parallel SelectMAP configuration interface was also tested successfully. Figure 67 shows a small VHDL application that was loaded into the FPGA to test if the programming of the configuration PROM was correct and whether the FPGA was configured after POR. Figure 67 shows the toggling of the LED's on the main board, which illustrates that the FPGA is configured correctly.

```

entity test is port(
    clk_25mhz: in std_logic;
    led       : out std_logic_vector(3 downto 0));
end test;

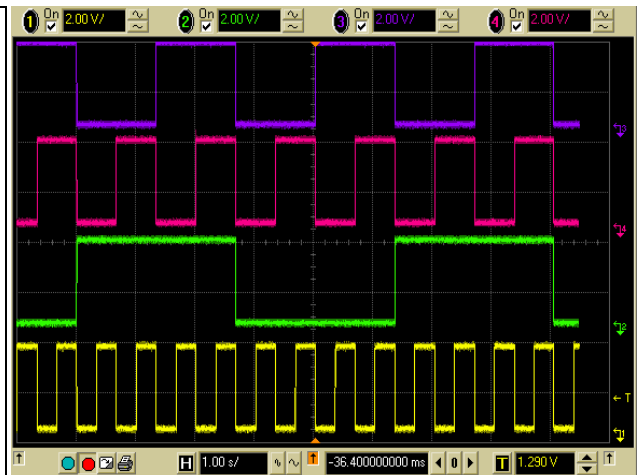
architecture Behavioral of test is
    signal clk_count : std_logic_Vector(26 downto 0);
begin

    process(clk_25mhz)
    begin
        if(rising_edge(clk_25mhz)) then
            clk_count <= clk_count + 1;
        end if;
    end process;

    led <= clk_count(26 downto 23);

end Behavioral;

```



**Figure 67. VHDL test application**

## Section 4.5.2 Power supply

The PSU PCB was tested by the manufacturer before delivery. When the board was received, all power supply and ground pins were tested for short circuit, connection and correct pin location. Figure 68 shows the assembled board.



**Figure 68. Top and bottom view of the power supply board**

One of the four DC-DC converters was built and tested before continuing with the other three. Each DC-DC converter was also tested by driving a resistive load at high current, to

establish if it can deliver the rated current and also to determine if there are any thermal issues on the board or components. Figure 69 shows the voltage ripple of each DC-DC converter under load. Xilinx specifies that the maximum amount of power supply noise or ripple must not exceed  $\pm 5\%$ . From Figure 69 it is evident that the ripple for each of the four system voltages are:

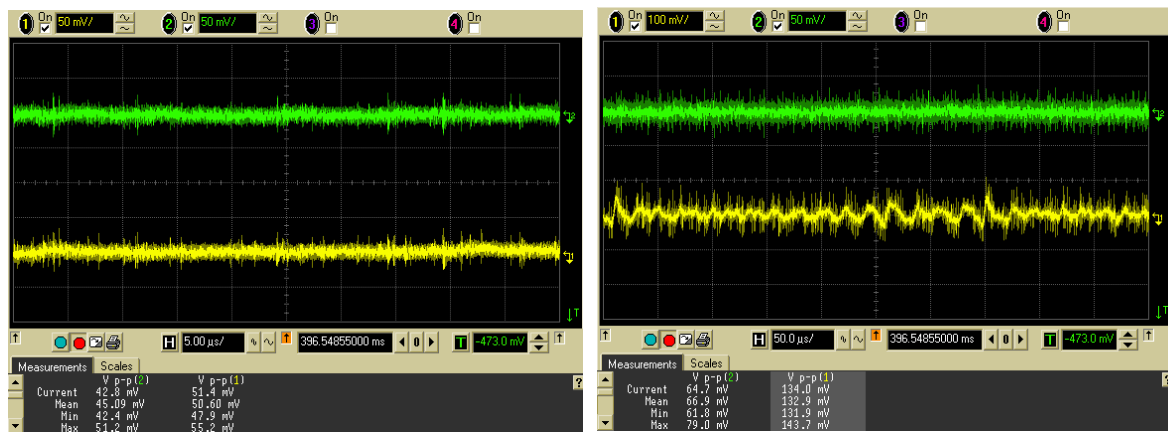
1.2V at 3.75% ripple

2.5V at 2.02% ripple

3.3V at 2.02% ripple

5V at 2.66% ripple

Thus the ripple voltage is well within Xilinx specifications.



**Figure 69. Supply voltage ripple. Left: 2.5V (yellow) & 1.2V (green). Right: 5V (yellow) & 3.3V (green)**

After this, the intelligent power switch was tested. This circuit is responsible for hot switching the system from internal battery to external supply and back again. When the external supply is connected the battery charging cycle must also start. The automatic switchover did not work due to an incorrect supplied relay. This relay was removed and replaced with a manual switch. Once this was working the charging circuit was tested. The system is connected to two 1800 mah Lithium Polymer batteries. The switching charger charges at a maximum rate of 2A and takes about 30 minutes to charge the two batteries. Most of the system debugging was done while running on battery power so the charging was constantly tested.

### Section 4.5.3 Peripheral board

The PSU PCB was tested by the manufacturer before delivery. When the board was received all power supply and ground pins were tested for short circuit, connection and correct pin location. Figure 70 shows the assembled peripheral board.



Figure 70. Top and bottom view of the peripheral board

Before testing the H-bridge controllers the current sensing resistors were unsoldered to remove the risk of destroying the Mosfets during a short circuit condition. At first a PWM signal was output to each Mosfet to test the PCB tracks, connectors and Mosfet drivers. After this the H-Bridge controller (see H-bridge controller in Section 4.1.4.1 ) was enabled to drive the full H-bridge. Each Mosfet gate signal was measured with an oscilloscope to ensure that there are no short-circuits. Once all these tests passed the current sensing resistor was added to complete the motor driving circuit and continue the test with the motor.

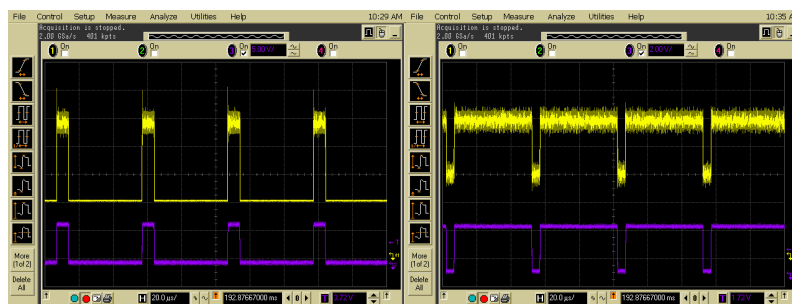


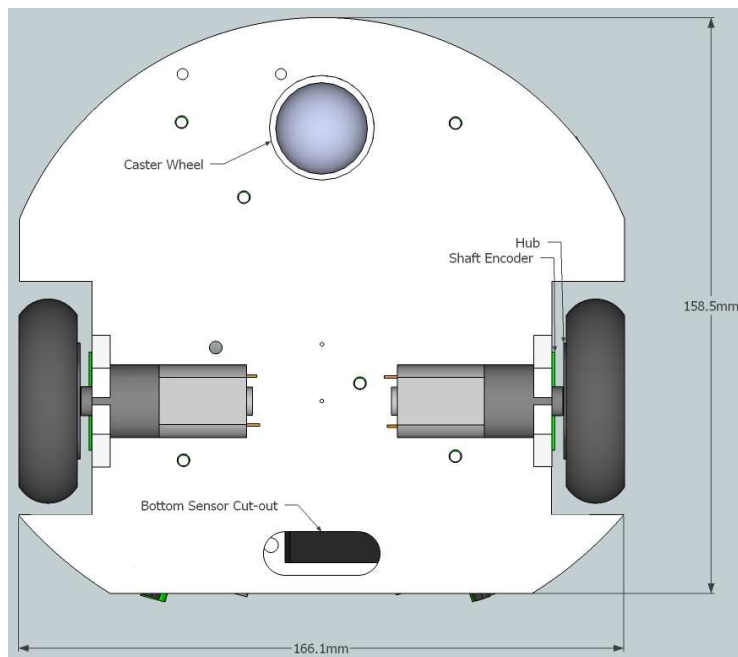
Figure 71. H-bridge waveforms

Figure 71 shows typical waveforms for the H-bridge connected to the main driving motor. The yellow signal is measured across the motor while the purple signal is the PWM applied to the lower Mosfet. On the left the motor is running at low speed while on the right it is

running at almost maximum speed. Very little voltage spikes are observed on the motor waves which is a sign of good board layout and design. A multilayer varistor is included on the board across the motor terminals to clamp down voltage spikes.

## Chapter 5 Robot Construction

The robot chassis is designed to accommodate two DC motors and one caster wheel. The DC motors are arranged in a dual differential drive configuration with the two driving wheels and the caster wheel lying in a circle. With all three wheels in a circle the robot has the ability to rotate around its own axis without any positional translation. The diameter of the circle is 150.2 mm (see Figure 72).



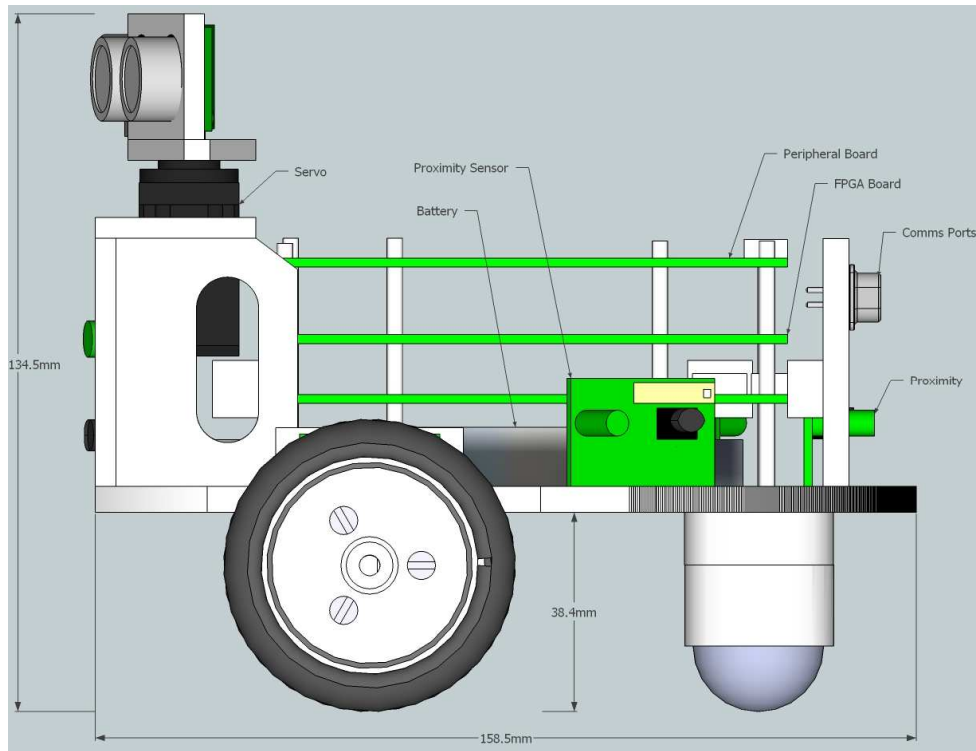
**Figure 72. Robot bottom view**

The caster wheel must have a similar height in comparison to the main driving wheels. This is to ensure that the robot stands almost level on a flat surface. The caster wheel should be able to spin and swivel evenly to ensure that it does not impede the robot's movement. Since the caster wheel provides only support and not traction, it should be constructed from hard material which will reduce friction. A ball caster (Figure 73) forms an ideal omni directional caster.

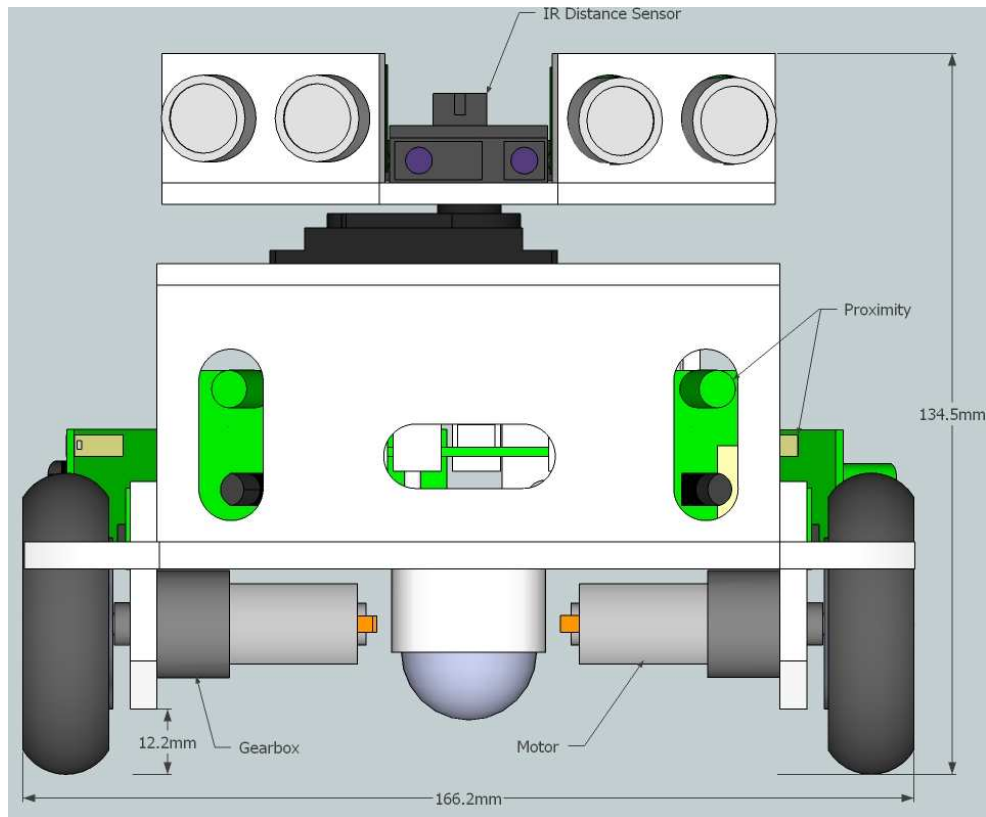




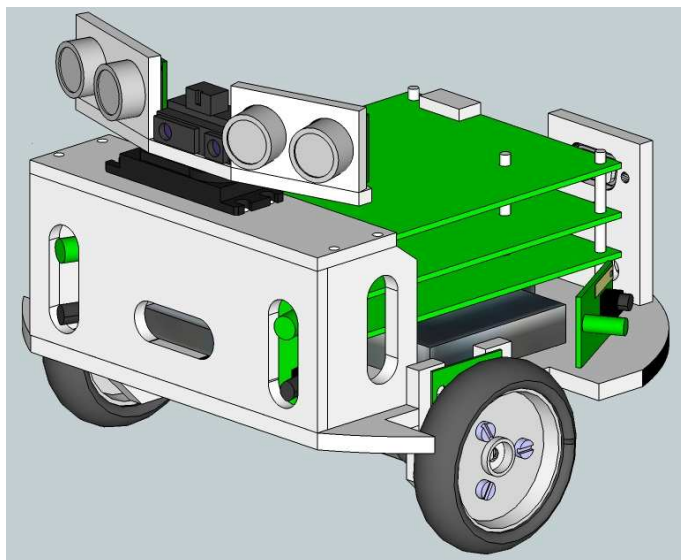
**Figure 73. Ball caster**



**Figure 74. Robot side view**



**Figure 75. Robot front view**



**Figure 76. Robot Perspective view**

The mechanical design for ASRA was done in 3D CAD software to ensure proper alignment of the wheels, caster wheel and correct placement of mounting holes. This also assists the placement of the shaft encoder PCB, and the placement of the ring magnet in the hub.

According to this design the hub was turned from aluminium on a lathe, and all the body parts were laser cut from Perspex. Because of the precision of the design and the slotted-mounting-holes on the shaft-encode PCB, it was easy adjusting the shaft-encode PCB to the optimal position. The robot base and motor mountings was made from 5mm Perspex while on the other components 3mm was used. Figure 74, Figure 75 and Figure 76 shows the final design for ASRA.

Each of the two ultrasonic sensor pairs is placed at a 15 degree offset, to ensure slight overlapping in there field of view. Having two sensors, reduce the scan time when applying the advanced sonar technique discussed in Section 4.4.2 .

The LCD display at the back of the robot is used as a debug aid. The display was salvaged from an old Nokia phone and utilise a SPI interface. It is not intended to be present on all robotic agents.

The first ASRA prototype is shown in Figure 77. It contains only two 3.7V 1800mAh Lithium Polymer batteries connected in series to supply 7.4V at 1800mAh. The mechanical design makes provision for four batteries, delivering a maximum of 7.4V at 3600mAh.

The small panel at the back of the robot next to the LCD display contains 2 RS232 serial ports (on the DB9 connector) of which one is used to debug and load code through the GDB-stub application. This panel also contains a power on switch and a hardware reset push button.

For now only the two front IR proximity sensors are installed, but the mechanical design also provides for a third sensor at the front, pointing towards the ground for sensing steps. It is also possible to install multiple IR proximity sensors around the robot enabling it to do proximity detection on any side.

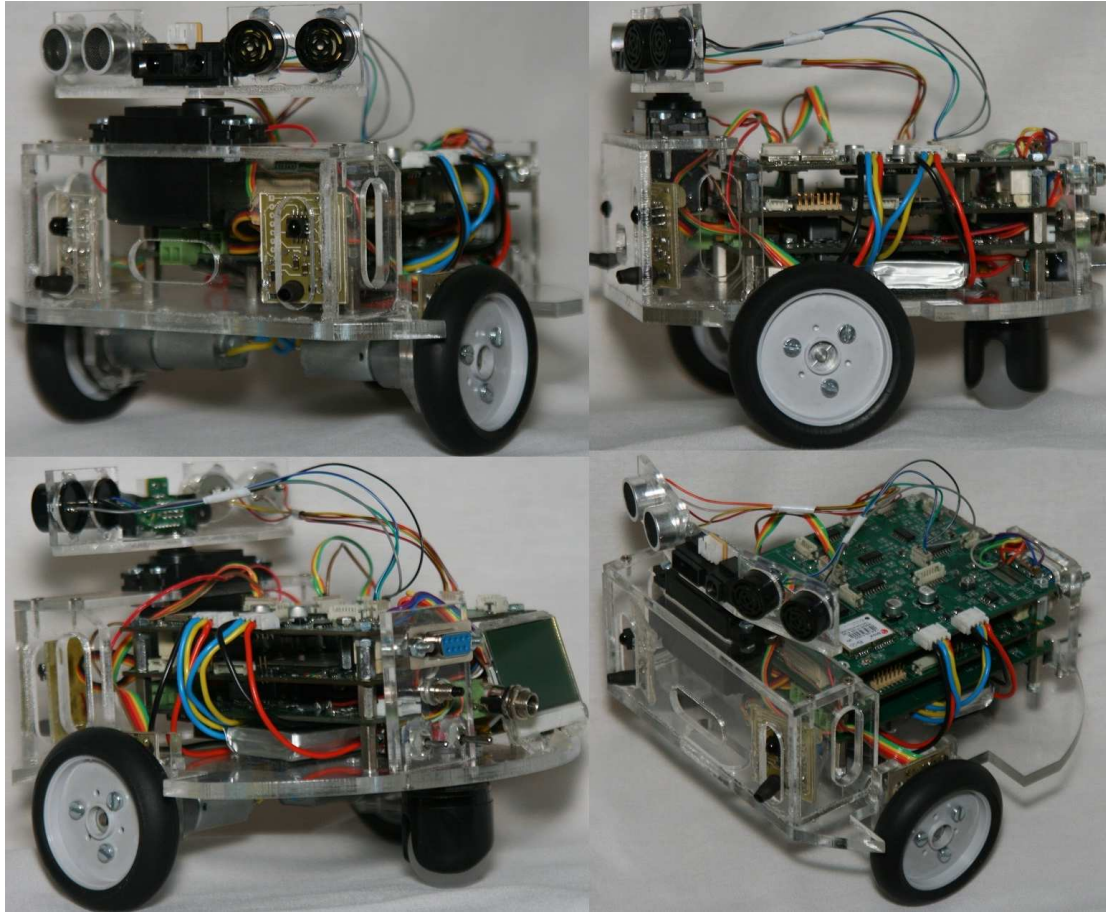


Figure 77. Constructed robotic agent

## Section 5.1 Motor

DC motors are commonly used in the toy and robot hobby market. It is easy to obtain a reasonable quality motor at a low price and for this reason a DC motor is used for the robot propulsion. The motors consumes the most energy in the system and it is there for important to run them at there optimal point. This section explains the math behind determining this point and at the end it shows the motor torque and power graph.

Motor torque is specified in Nm.

The electrical power to the motor in Figure 78 is:  $P_e = V \cdot I$   $[J / s = W]$

Mechanical power:  $P_m = T \cdot \omega$

$$P_m = P_e \cdot \eta \quad \text{Where } \eta \text{ is the percentage efficiency of the motor}$$

The motor rotor coil is an inductor with an induced voltage across it of:  $v = L \cdot \frac{di}{dt}$

This induced voltage opposes the applied driving voltage (V). The faster the motor turns the more often the rotor coils move through the magnetic field of the permanent magnets thereby inducing a higher apposing voltage. The apposing voltage limits the current through the resistance (R) of the motor which reduces the flux created around the conductor causing a drop in torque.

From Figure 78:  $V = IR + e$       Where e is the back EMF of the motor

When the motor is not turning  $e = 0$ . Thus the starting current of the motor is:

$$I_s = \frac{V}{R}$$

When the motor is rotating:  $e = k_e \cdot \omega$  where  $k_e$  is the back EMF constant

Thus we can write:  $V = I \cdot R + k_e \cdot \omega$  during rotation.

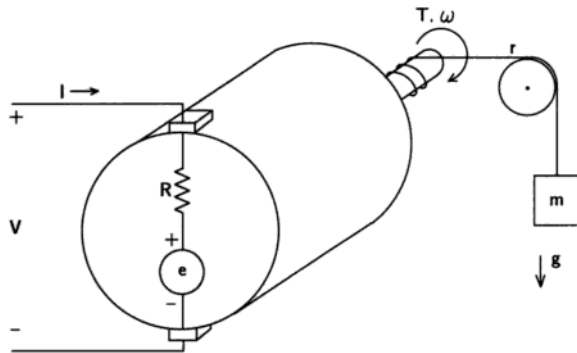


Figure 78. DC motor model [37:205]

The torque increases linearly with a proportional constant  $k_t$  known as the torque constant.

$$T = k_t \cdot I$$

$$\text{Thus we can write: } V = \frac{T \cdot R}{k_t} + k_e \cdot \omega \quad (1)$$

The mechanical power at the shaft is equal to the electrical input power minus the electrical losses.

$$P_m = P_e - I^2 R$$

$$\therefore T \cdot \omega = V \cdot I - I^2 \cdot R$$

$$\therefore k_t \cdot I \cdot \omega = (I \cdot R + k_e \cdot \omega) \cdot I - I^2 \cdot R \quad (\text{Substitute T \& V})$$

$$\therefore k_t = k_e = k$$

From (1):  $V = \frac{T \cdot R}{k} + k \cdot \omega$  (2)

$$\boxed{\therefore \omega = -\frac{R \cdot T}{k^2} + \frac{V}{k}} \quad (3)$$

$$\boxed{P_m = T \cdot \omega = -\frac{R \cdot T^2}{k^2} + \frac{V \cdot T}{k}} \quad (4)$$

The no load speed ( $N_0$ ) of a motor is when the torque is null. From (2):  $\boxed{\omega_n = \frac{V}{k}}$

Where  $\omega_n = \left( \frac{N_0}{60} \right) \cdot 2\pi$  [rad/s]

The stall torque ( $T_s$ ) of a motor is when the speed is null. From (2):  $\boxed{T_s = \frac{k \cdot V}{R}}$

The torque at which maximum mechanical power occurs can be found by taking the derivative of the power with respect to the torque and setting it equal to 0.

$$\frac{dP_m}{dT} = 0 = \frac{-2R \cdot T}{k^2} + \frac{V}{k}$$

$$\therefore T_{P_{\max}} = \frac{1}{2} \cdot \frac{k \cdot V}{R} = \frac{1}{2} \cdot T_s$$

Thus the speed at maximum power point (from (3)):

$$\omega_{p_{\max}} = \frac{-R}{k^2} \cdot \frac{k \cdot V}{2R} + \frac{V}{k} = \frac{V}{2k} = \frac{1}{2} \cdot \omega_n \quad (5)$$

The maximum mechanical power then simply is:

$$P_{\max} = T_{p_{\max}} \cdot \omega_{p_{\max}} = \frac{1}{2} \cdot T_s \cdot \frac{1}{2} \cdot \omega_n = \frac{1}{4} \omega_n \cdot T_s \quad (6)$$

Maximum efficiency cannot be achieved at maximum power. We want to operate the motor at maximum efficiency. Maximum efficiency is related to the stall and no load current.

$$\eta_{\max} = \left( 1 - \sqrt{\frac{I_0}{I_s}} \right)^2 \quad (\text{see [37:210]}) \quad (7)$$

If the motor is run at a lower voltage, the speed and the torque are reduced. Another way to decrease the speed of the motor is to use a gearbox. By gearing down a motor by a factor of 2, the no load speed is cut in half but the stalling torque is doubled.

Torque after gearbox:  $T_{\text{gear}} = P_m \cdot N_{\text{gearbox}} \cdot \eta_{\text{gearbox}}$

Operating voltage	6V
No load speed $N_0$	90 rpm
Free run current $I_0$	250 mA
Stall current $I_s$	3.3A
Stall torque $T_s$ @ 6V	0.8614 Nm
Weight	52.45 g
Gear ratio	154:1

**Table 27. Geared motor specifications**

After considering numerous low cost DC motors for driving the wheels, it was decided to use the metal-gear motor from Pololu Robotics (see Figure 79). The cost of a motor is 20 USD and it uses metal gears enclosed in an aluminium casing. Because the gearbox is enclosed it is

ideal for driving wheels which is close to the ground and exposed to dust. It is also possible to put lubrication in the gearbox thereby increasing the efficiency of it.



**Figure 79. Geared motor**

From Table 27 the following calculations can be made.

$$\omega_n = \left( \frac{N_0}{60} \right) \cdot 2\pi = 9.425 \quad [\text{rad/s}]$$

$$\omega_n = \frac{V}{k} \Rightarrow k = \frac{V}{\omega_n} = 0.6366$$

$$T_s = \frac{k \cdot V}{R} \Rightarrow R = \frac{k \cdot V}{T_s} = 4.4343 \quad [\text{ohm}]$$

From (7)  $\eta_{\max} = 52.53\%$

$$\text{From (6)} \quad P_m = \frac{1}{4} \omega_n \cdot T = 2.03 \quad [\text{W}]$$

Now let's write the equations for the torque speed curve.

From (3) and the equations for  $T_s$  and  $\omega_n$  we can write the following:

$$T(\omega) = T_s - \frac{\omega T_s}{\omega_n} = 0.8614 - 0.0914\omega \quad [\text{Nm}] \quad (8)$$

Or

$$\omega(T) = \frac{(T_s - T)\omega_n}{T_s} = 10.94(0.8614 - T) = 9.425 - 10.94T \quad [\text{rad/s}] \quad (9)$$

From (5) and (6) we see that the maximum mechanical power is:

$$P_{\max} = \frac{1}{4} \omega_n \cdot T_s = 2.03 \quad [\text{W}]$$



Which occurs at:  $\omega_{p_{\max}} = \frac{1}{2} \cdot \omega_n = 4.713$  [rad/s]

By substituting (8) and (9) into  $P_m = T \cdot \omega$  we obtain the power curve equation.

$$\begin{aligned} P_m(\omega) &= -\left(\frac{T_s}{\omega_n}\right)\omega^2 + T_s\omega = -0.0914\omega^2 + 0.8614\omega \\ P_m(T) &= -\left(\frac{\omega_n}{T_s}\right)T^2 + \omega_n T = -10.94T^2 + 9.425T \end{aligned} \quad (10)$$

The calculated power and torque are plotted in Figure 81.

### Section 5.1.1 Experimental results

The winding and brush resistance ( $R_a$ ) can be measured with a normal multimeter but a more accurate measurement is obtained, by lightly powering the motor and measuring the voltage and current while the shaft is not turning (no back emf).

With the motor connected to a lab power supply, slowly begin to increase the supply voltage from 0 till the shaft start to turn slightly. Decrease the supply voltage to stop rotation and measure the voltage and current.

V [mV]	I [mA]	R = V/I [ohm]
340	90	3.778
216	59.7	3.618

**Table 28. Motor resistance measurements**

Averaging Table 28 readings gives:  $R_a = 3.698$  ohms

Figure 80 shows the no load current and speed measurements and the motor stall current measurements. To make the speed measurements, the shaft encoder in Section 4.4.3 was used. Because we are using the linear approximation model for the motor, it is possible to do a straight line extrapolation on the stall current graph.

At 6 V the stall current is:  $I_s = 2.25$ A.

From the same figure the no load speed at 6 V is:  $N_0 = 79$  rpm thus  $\omega_n = 8.273$

The no load current is:  $I_0 = 175$  mA

$$\omega_n = \frac{V}{k} \Rightarrow k = 0.7252$$

$$T_s = \frac{k \cdot V}{R} = 1.1767$$

$$\omega_{p_{\max}} = \frac{1}{2} \cdot \omega_n = 4.1365 \text{ rad/s} \Rightarrow N = 39.5 \text{ rpm}$$

From equation 8 we can deduce the following torque equation:

$$T(\omega) = T_s - \frac{\omega T_s}{\omega_n} = 1.1767 - 0.1422\omega \quad \text{Nm}$$

And from equation 10 we can deduce the following mechanical power equation:

$$P_m(\omega) = -\left(\frac{T_s}{\omega_n}\right)\omega^2 + T_s\omega = -0.1422\omega^2 + 1.1767\omega \quad \text{W}$$

Figure 81 shows the graph for these two equations. The purple dotted rectangular region represents the maximum mechanical power measured, and the orange rectangular region represents the maximum mechanical power calculated.

Incorrect manufacturer data can be accounted for the difference between the calculated torque and power graph and the measured one. It should be taken into account that this motor is low cost and was sourced from the hobbyist robotic market. The manufacturer data is very limited in comparison to an expensive motor.

For this research we will work with the measured graph (purple and green lines in Figure 81).

From this graph and equation 5 we determine that the optimal point is at:

$$\omega_{p_{\max}} = \frac{1}{2} \cdot \omega_n = 4.14 \text{ [rad/s]}$$

$$P_m = 2.434 \quad \text{[W]}$$

$$T_m = 0.59 \quad \text{[Nm]}$$

The wheel speed is:  $v = r \cdot \omega = 0.12$  [m/s]

The radius of the robot wheel is:  $r = 29$  [mm]

Robot weight:  $m = 750$  [g]

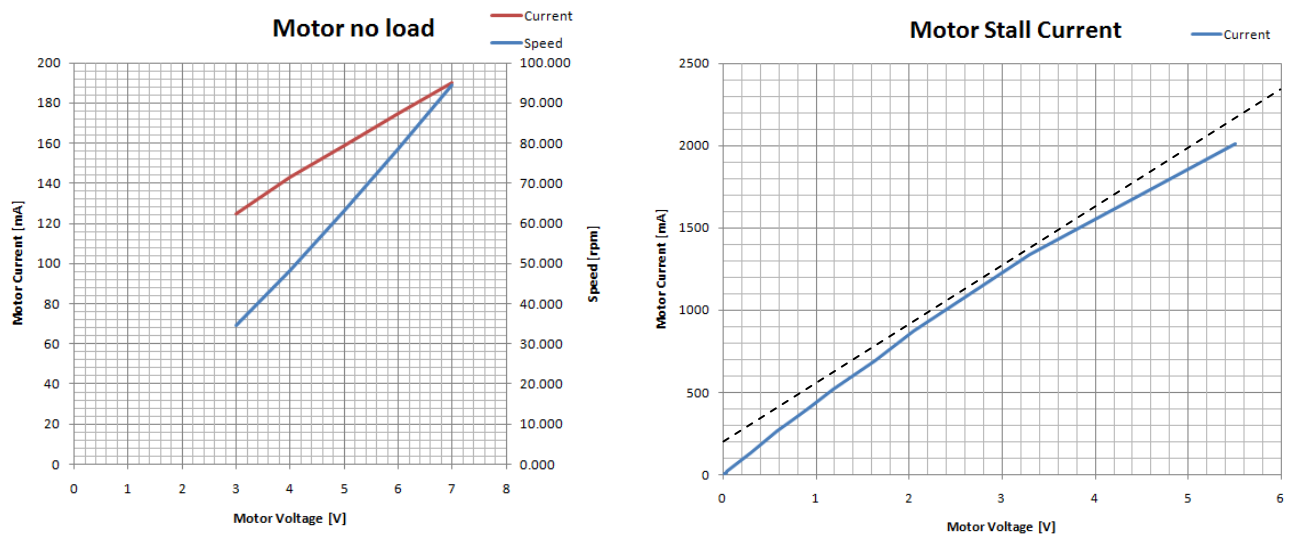


Figure 80. Motor no load and stall measurements

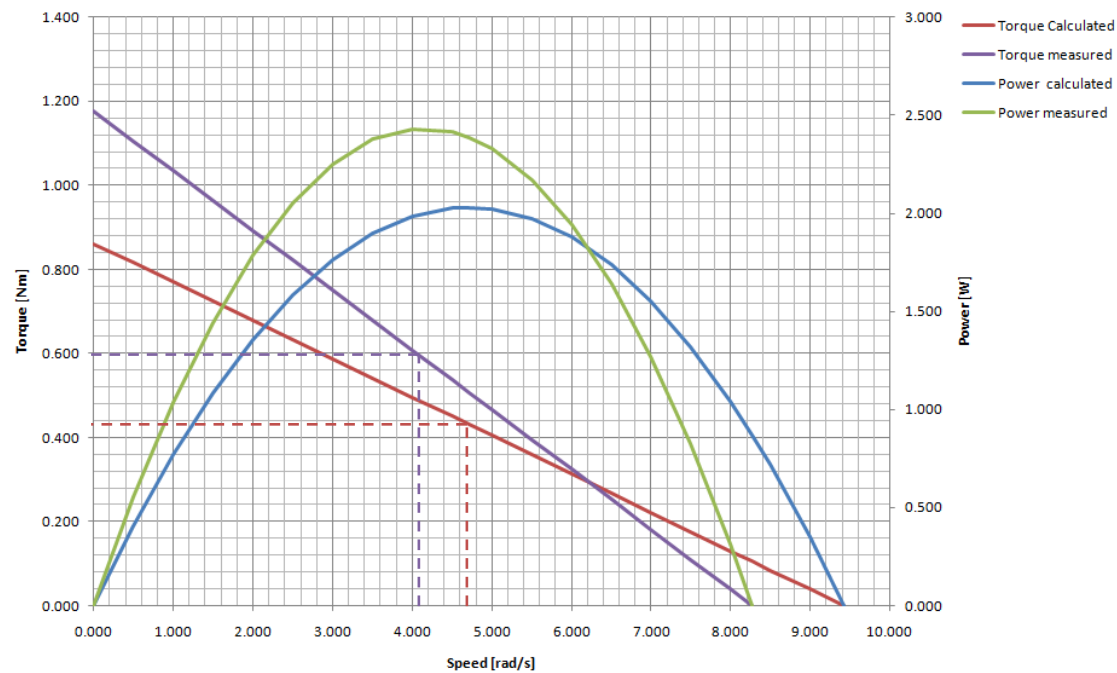


Figure 81. Motor power and torque graphs

***Assumptions:***

Static friction for rubber on dry concrete  $\mu_s = 1$

$\mu_k = 0.005$  (Typical BMX rubber wheel)

We are working on a horizontal flat surface.

No adjustments for velocity in kinetic friction calculations

Thus the static friction force is:  $F_s = \mu_s \times N = \mu_s \times m \times g = 7.355$  [N]

And, the kinetic friction force is:  $F_k = \mu_k \times m \times g = 0.037$  [N]

The wheel torque to overcome the friction:

$$T_s = r \times F_s = 0.213 \quad [\text{Nm}]$$

$$T_k = r \times F_k = 0.001 \quad [\text{Nm}]$$

At the optimal point the motor delivers 0.59 Nm torque, which is adequate to overcome the static friction.

## Chapter 6 Conclusion and future work

The aim of this research was to develop a robot platform for swarm intelligence research. The first three chapters described the problem, and theory driving the design. Chapter four and five contains the bulk of the work done. The original aim was optimistic in its goals, time estimation and hope that towards the end of this research, there would be enough time to build a second agent and conduct some basic experiments related to swarm interaction. Due to project complexities there was not enough time to conduct experiments other than the ones presented in this report. The remainder of work is left for future research.

### ***Section 6.1 Original contribution***

The unique requirements of the project made it difficult to buy standard off-the-shelf hardware. A study was made of available technologies and none could be found that met the requirements. Most of the standard products are physically big and was not designed for low power battery applications. Almost all the available hardware use DDR RAM, which is renowned for its high power consumption. They also include Ethernet and various other peripherals.

The hardware design utilises some of the latest technologies on the market which ensure a flexible design with the lowest power consumption, a feature seldom seen on high-end FPGA boards. Only the core peripherals were included to ensure that there are no redundant or unused devices which can draw additional power. The main board has a general purpose bus and a peripheral bus. There are also mounting holes around the fine pitch bus connectors. Together this forms a stackable expansion bus, similar but smaller than the PC104 standard. Thus, additional functionality through peripheral devices can be added. The reconfigurable FPGA with soft processor core enhances this ability.

The main board was designed with partial self reconfiguration in mind. Pin allocation was done in such a way that the core peripherals, the instruction and data bus for the soft-core are within the fixed area allocated for the soft-core. This eliminates the need for additional routing traces inside the FPGA going from the fixed area through the reconfigurable area to the external pins. The fixed area was also placed at a location which ensures the maximum

resources available (BRAM and DSP48 blocks) for the soft-core. The remaining section of the FPGA was divided into almost homogeneous reconfigurable blocks, each containing external pins going to the expansion bus connector. Except for the FPGA specific design tools (from Xilinx) all tools and source code used falls under the GPL license agreement. This makes ASRA the ideal research platform.

The intelligent power management peripherals combined with partial reconfiguration forms a powerful combination in assessing and addressing the growing power needs of complex swarm robotic research. Highly efficient DC-DC converters ensure multiple voltages available to interface to most sensors and actuators. Battery charging and management is handled by ASRA and only an external voltage of 12V or greater is needed. ASRA will disconnect once the battery is fully charged.

Additional sensors were developed for ASRA. This includes proximity detection and shaft encoding. The proximity sensors form a low cost infrared sensor network around the robot which replaces the often-used tactile/micro switch mechanical proximity detectors. There are no moving mechanical components which improve reliability and simplify the mechanical design. The construction cost of the shaft encoder is also low and its resolution is very good.

The mechanical design was specifically made for this application. Each board, sensor and actuator was model in 3-D and included in the design. This ensured optimal usage of space and proper alignment of components and mountings. IT also made it easy to integrate the designed shaft encoders into the system

## ***Section 6.2 Known issues***

Unfortunately there are some issues on different aspects of the system. This is the first iteration of the design cycle and it is unrealistic to expect everything to work flawlessly. This section lists all the known electronic design and construction issues, as well as system issues and shortfalls.

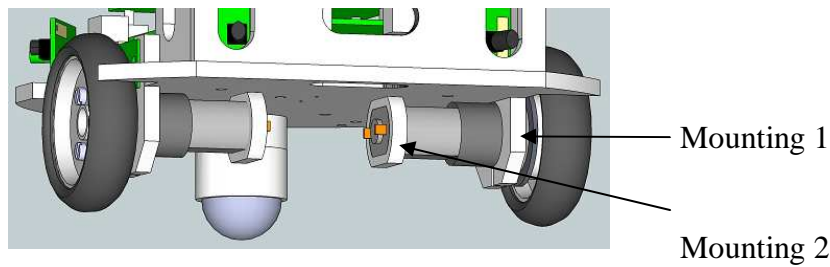
## ***Electronic hardware***

The main board contains status LED's to indicate when the FPGA reconfiguration cycle is done as well as the reset state of the FPGA. Both these status LED's are driven by a dual bipolar transistor (Q1 on sheet 4 of the schematic). The base on both of these transistors does not contain a series resistor. A series 1K ohm resistor must be added to avoid the transistor pulling the FPGA's reset and configuration line to 0.6V ( $V_{base}$ ).

To lower the risk in component sourcing multiple flash devices was taken into consideration when the flash device footprint was created. The created footprint supported Numonyx, ST and Spansion devices. Because of this ball G2 was tied to ground to cater for parts that used address line zero. On the prototype a Numonyx device was populated which shows ball G2 as reserved. This caused the component to malfunction and draw a lot of current. The author had to send the board back to the production company to remove the flash device and populate a new device of which ball G2 was removed. Thus, when populating the flash device with a Numonyx component ball G2 must be removed.

## ***Mechanical***

The most important aspect of ASRA's construction is the alignment of the two motors and the caster wheel. As stated previously these components must be in a circle. With the current design it is difficult to align them, and once aligned it is uncertain if they will stay in the correct position. The problem lies with the one, 5mm motor mount for each motor. For the first prototype a jig was made to properly align the motors, and ensure the correct distance between the two tyres. Once aligned the two motors were glued on to an aluminium strip and inserted into the two motor mountings. This made the construction sturdy. For future units a design change was made to resolve this issue. Figure 82 shows the changes in the form of a second set of motor mountings.



**Figure 82. Additional motor mountings**

The mounting holes for the shaft encoders were made too big. The screws protrude above the PCB and can easily touch the ring magnet. The sensing distance is adequate to avoid contact but special care must be taken when calibrating this sensor. For future use the CAD files were updated to allow the use of smaller screws.

At the back of ASRA there is a panel containing a DB9 connector and some switches. The location of the panel is too close to the main PCB stack and it was difficult install the switches. For the next revisions this panel must be moved to the back further away from the PCB stack.

### ***Shortfalls***

The main motor's torque is adequate but the speed is too slow. In Section 5.1 we saw that the optimal load point for this motor is at 45 rpm (half of the maximum speed). ASRA can control the speed of the motor, and a higher speed motor will give more flexibility and show faster convergence when doing swarm experiments. At the moment ASRA is using the motors to its maximum speed. This is inefficient as we are not using the motor at its optimal point, thereby wasting electric power.

The power supply charging circuit only caters for two Lithium Polymer/Ion batteries connected in series. The highest rated battery easily obtained was 1800mAh. This is more than adequate for the electronics but when including higher speed driving motors, this is not enough. It might be a good idea to adapt the system to charge four batteries. This can be accomplished in one of two ways: 1) Connect the first set of batteries through a relay to the charger and once fully charged disconnect and reconnect the second set. The onboard microcontroller will have to manage this process. In the current design the charger supplies



the micro controller with charge status, so this is a feasible solution. 2) Include an additional PCB between the charger and the four batteries which contains electronics to do load balancing to all four batteries. This solution is by far the most elegant, and the radio controlled model toy industry contains numerous low cost devices that will work here.

### **Section 6.3 Future work**

The electronic and mechanical design of ASRA was underestimated and it took longer to develop it. The core components tested were propulsion, battery and power management, processing (the brain) and some of the sensors. Peripheral drivers were developed for all the tested components, and all testing was conducted with the FreeRTOS port for this project.

Although the infrared proximity sensor was tested thoroughly for indoor and outdoor use, it was not completely integrated into ASRA. It is envisaged that ASRA will contain at least eight of these sensors to form infrared proximity radar around the robot. All these sensors will use a common I2C bus, although the system contains enough general purpose I/O to use each sensor discreetly. The testing and integration of this sensor network into the higher levels of behaviour is left for future research.

Accurate navigation is extremely important for autonomous operation. Therefore an accurate low cost shaft encoder was developed to reduce the error when doing dead reckoning. It is however necessary to rely on frequent fixed navigation points to zero the accumulated dead reckoning errors. For this reason a GPS receiver was included in the system on the peripheral board. The intention was to use this to obtain these fixed points. To improve the navigation system even further the peripheral expansion bus and A2D ports can be used to interface to an IMU module containing a 3-axis gyroscope, 3-axis accelerometer, barometric pressure sensor and temperature sensor. The integration of the GPS into the navigation system and the possible inclusion of a 6-DOF IMU unit are left for further research. It should however be noted that without these components and there integration, autonomous navigation will be very difficult.

Chapter 2 showed the literature study made on swarm robotics. In this chapter basic behaviour and a layered control system was discussed. It further showed the implementation thereof. Basic behaviour and the implementation of the control layers are research specific and are left to the researcher using ASRA as his or her research platform.

## References

- [1] Mark Russel Edelen, “Swarm Intelligence and Stigmergy: Robotic Implementation of Foraging Behaviour”, Master of science, University of Maryland, 2003.
- [2] Rodney A. Brooks, “A Robust layered control system for a mobile robot”, A.I Memo 864. Massachusetts Institute of Technology, September 1985.
- [3] Maja J. Matarić, “Interaction and Intelligent Behaviour”, Ph.D. dissertation, Massachusetts Institute of Technology USA, 1994.
- [4] Nicholas Peter Sedcole, “Reconfigurable Platform-Based Design in FPGAs for Video Image Processing”, Ph.D. dissertation, University of London, United Kingdom, January 2006.
- [5] Maja J. Matarić, “Using Communication to Reduce Locality in Destributed Multi-Agent Learning”, Journal of Experimental & Theoretical Artificial Intelligence, Volume 10, Issue 3, pp. 357-369, July 1998.
- [6] Maja J. Matarić, “Behaviour-Based Control: Examples from Navigation, Learning, and Group Behaviour”, Journal of Experimental & Theoretical Artificial Intelligence, Volume 9, Issue 2 & 3, pp. 323-336, April 1997.
- [7] Maja J. Matarić, “Behaviour-Based Control: Main Properties and Implications”, IEEE International Conference on Robotics and Automation, Workshop on Architectures for Intelligent Control Systems, pp. 46-54, 1992.
- [8] Maja J. Matarić, “Behaviour-Based Robotics”, MIT Encyclopedia of Cognitive Sciences, Robert A. Wilson and Frank C. Keil, eds., MIT Press, pp. 74-77, April 1999.

- [9] Maja J. Matarić, “Behaviour-Based Robotics as a Tool for Synthesis of Artificial Behaviour and Analysis of Natural Behavior”, Trends in Cognitive Sciences, Volume 2, Issue 3, pp. 82-86, 1 March 1998.
- [10] Maja J Matarić, “Designing and Understanding Adaptive Group Behaviour”, Adaptive Behavior journal, Volume 4, pp. 51-80, 1995.
- [11] Maja J. Matarić, “Issues and Approaches in the Design of Collective Autonomous Agents”, Robotics and Autonomous Systems Journal, Volume 16, pp. 321-331, 1994.
- [12] Kehuai Wu, “Reconfigurable Architectures: from Physical Implementation to Dynamic Behaviour Modelling”, Ph.D. dissertation, Technical University of Denmark, Denmark, 2007.
- [13] K. Paulsson, M. Hübner and J. Becker, “On-Line Optimization of FPGA Power-Dissipation by Exploiting Run-time Adaptation of Communication Primitives”, Proceedings of the 19th annual symposium on Integrated circuits and systems design, pp. 173-178, 2006.
- [14] Barry Brian Werger, “Cooperation without Deliberation: A Minimal Behaviour-based Approach to Multi-robot Teams”, Journal, Artificial Intelligence - Special issue on Robocop: the first step, Volume 110, Issue 2, pp. 293-320, June 1999.
- [15] G. Sutter and E. Boemo, “Experiments in low power FPGA design”, School of Engineering, Universidad Autónoma de Madrid, 2007.
- [16] Abhishek Mitra, Zhi Guo, Anirban Banerjee and Walid Najjar, “Dynamic Co-Processor Architecture for Software Acceleration on CSoCs”, Computer Design, pp. 127-133, October 2006.
- [17] Texas Instruments , Tips for successful power-up of today’s high-

performance FPGAs Jeff Falin. Appl. Note 079, 2004.

- [18] Mateusz Majer, Ali Ahmadiania, Christophe Bobda and Jürgen Teich, “A Flexible Reconfiguration Manager for the Erlangen Slot Machine”, In Dynamically Reconfigurable Systems Workshop, pp. 183-194, 2006.
- [19] A. Gayasen, K. Lee, N. Vijaykrishnan, M. Kandemir, M. J. Irwin and T. Tuan, “A Dual-Vdd Low Power FPGA Architecture”, In Proceedings of International Conference on Field Programmable Logic and Applications, pp. 145-157, 2004.
- [20] Yuanlin Lu, “Power and performance optimization of static CMOS circuits with process variation”, Ph.D. dissertation, Auburn University Alabama, 2007.
- [21] S. Hackwood and G. Beni, "Self-organizing sensors by deterministic annealing," presented at IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 1177-1183, 1992.
- [22] J. L. Deneubour, G. Theraula, and R. Beckers, “Swarm-made architectures” Presented at European Conference on Artificial Life, pp. 123-133, 1992.
- [23] Eric Bonabeau, Marco Dorigo and Guy Theraulaz, “Swarm Intelligence. From Natural to Artificial Systems”, Santa Fe Institute Studies in the Sciences of Complexity Proceedings, Oxford University Press, 1999.
- [24] S. Camazine, J. L. Deneubourg, N. R. Franks, J. Sneyd, G. Theraulaz, and E. Bonabeau, “Self-Organization in Biological Systems”, Princeton University Press, 2001.
- [25] E. Bonabeau, G. Theraulaz, J. L. Deneubourg, S. Aron, and S. Camazine,

- “Selforganization in social insects”, Trends in Ecology and Evolution, vol. 12, pp. 188-193, 1997.
- [26] T. Arai, E. Pagello, and L. E. Parker, “Advances in multirobot systems”, IEEE Transactions on Robotics and Automation, vol. 18, pp. 655-661, 2002.
  - [27] Y. Cao, A. Fukunaga, and A. Kahng, “Cooperative mobile robotics: Antecedents and directions”, Autonomous Robots, vol. 4, pp. 1-23, 1997.
  - [28] A. Drogoul and J. Ferber, “From Tom Thumb to the Dockers: Some experiments with foraging robots”, Presented at Second International Conference on Simulation of Adaptive Behavior, Honolulu, HI, pp. 451-459, 1992.
  - [29] Xilinx, App Note DS123 (v2.15), Platform Flash In-System Programmable Configuration PROMs. July 07, 2008.
  - [30] Xilinx Virtex-4 FPGA User Guide. UG070 (v2.6) December 1, 2008.
  - [31] Xilinx Virtex-4 Data Sheet: DC and Switching Characteristics. DS302 (v3.1) December 11, 2007.
  - [32] Xilinx Platform Flash In-System Programmable Configuration PROMs. DS123 (v2.15) July 07, 2008.
  - [33] Xilinx Virtex-4 Configuration Guide. UG071 (v1.9) October 1, 2007.
  - [34] Xilinx Virtex-4 Data Sheet: DC and Switching Characteristics. DS302 (v3.1) December 11, 2007.
  - [35] Shannon Koh and Oliver Diessel, “Communications Infrastructure Generation for Modular FPGA Reconfiguration”, In Proc. IEEE Int. Conf. Field-Programmable Tech., pp. 321-324, 2006.

- [36] Phillip H. Jones, Young H. Cho and John W. Lockwood, “Dynamically Optimizing FPGA Applications by Monitoring Temperature and Workloads”, Proceedings of the 20th International Conference on VLSI Design held jointly with 6th International Conference: Embedded Systems, 2007.
- [37] J. L. Jone, A. M. Flynn and B. A. Seiger, “Mobile Robots second edition”, Published by A K Peters, Ltd., ISBN 1568810970, 9781568810973, 1999.
- [38] Microchip App. Note AN693. Understanding A/D Converter Performance Specifications. Steve Bowling.
- [39] Digi White paper, Wireless Mesh Networking ZigBee vs. DigiMesh.
- [40] SR04 Mobile Robot. David P. Anderson. Department of Geological Sciences, Southern Methodist University, SR04.pdf p 7, 1998.  
<http://www.geology.smu.edu/~dpa-www/robots/sr04/index.html>
- [41] Directed Sonar Sensing for Mobile Robot Navigation. John J. Leonard and Hugh F. Durrant-Whyte. The Springer International Series in Engineering and Computer Science, Vol. 175, 1992.